Auton Agent Multi-Agent Syst (2014) 28:558–604
DOI 10.1007/s10458-013-9232-2

BDD-versus SAT-based bounded model checking for the existential fragment of linear temporal logic with knowledge: algorithms and their performance

Artur Męski · Wojciech Penczek · Maciej Szreter · Bożena Woźna-Szcześniak · Andrzej Zbrzezny

Published online: 31 August 2013

Abstract The paper deals with symbolic approaches to bounded model checking (BMC) for the existential fragment of linear temporal logic extended with the epistemic component (ELTLK), interpreted over interleaved interpreted systems. Two translations of BMC for ELTLK to SAT and to operations on BDDs are presented. The translations have been implemented, tested, and compared with each other as well as with another tool on several benchmarks for MAS. Our experimental results reveal advantages and disadvantages of SAT-versus BDD-based BMC for ELTLK.

Keywords Bounded model checking · Binary decision diagrams · Propositional satisfiability problem (SAT) · Interpreted systems · Interleaved interpreted systems · Epistemic linear temporal logic

A. Męski · W. Penczek · M. Szreter
Institute of Computer Science, PAS, Jana Kazimierza 5, 01-237 Warszawa, Poland
e-mail: mszreter@ipipan.waw.pl

B. Woźna-Szcześniak (✉) · A. Zbrzezny
Jan Długosz University, IMCS, Armii Krajowej 13/15, 42-200 Czestochowa, Poland
e-mail: bwozna@gmail.com

A. Zbrzezny
e-mail: a.zbrzezny@ajd.czest.pl

W. Penczek
University of Natural Sciences and Humanities, II,
Sienkiewicza 51, 08-110 Siedlce, Poland
e-mail: penczek@ipipan.waw.pl

A. Męski
University of Łódź, FMCS, Banacha 22, 90-238 Lodz, Poland
e-mail: meski@ipipan.waw.pl

1 Introduction

Verification of multi-agent systems (MAS) is an actively developing field of research [7,8,14,24,25,30,47]. Several approaches based on model checking [12,48] have been put forward for the verification of MAS. Typically, they employ combinations of the epistemic logic with either branching [8,30,43] or linear time temporal logic [17,22,38]. Some approaches reduce the verification problem to the one for plain temporal logic [6,22], while others treat typical MAS modalities such as (distributed, common) knowledge as first-class citizens and introduce novel algorithms for them [38,43].

In an attempt to alleviate the state-space explosion problem (i.e., an exponential growth of the system state space with the number of the agents) two main approaches have been proposed based on combining bounded model checking (BMC) with symbolic verification using translations to either ordered binary decision diagrams (BDDs) [26] or propositional logic (SAT) [41]. However, the above approaches deal with the properties expressed in the existential fragment of CTLK (i.e., CTL extended with the existential epistemic components, called ECTLK) only. In the paper [46] a method for model checking LTLK formulae using BDDs is described, but it is not explained how it can be used for BMC.

In this paper we aim at completing the picture of applying the BMC-based symbolic verification to MAS by looking at the existential fragment of LTLK (i.e., LTL extended with the existential epistemic components, called ELTLK), interpreted over both the subclass of interpreted systems (IS) called interleaved interpreted systems (IIS) [31] and interpreted systems themselves. IIS are an asynchronous subclass of interpreted systems [16] in which only one action at a time is performed in a global transition. Our original contribution consists in defining the following four novel bounded model checking methods for ELTLK: the SAT-based BMC for IS and for IIS, and the BDD-based BMC for IS and for IIS. Moreover, we would like to point out that the proposed SAT-based BMC for ELTLK and for IS has never been defined and experimentally evaluated before. Next, both the presented BDD-based methods have been published earlier, but only in the informal proceedings of the LAM'2012 workshop.

All the proposed BMC methods have been implemented as prototype modules of VerICS [28], tested, and compared with each other as well as with MCK [17] on three well-known benchmarks for MAS: a (faulty) train controller system [21], a (faulty) generic pipeline paradigm [40], and the dining cryptographers [10]. Our experimental results reveal not only advantages and disadvantages of ELTLK SAT- versus BDD-based BMC for MAS that are consistent with comparisons for temporal logics [9,13], but also show two novel findings. Namely, IIS semantics can improve the practical applicability of BMC, and the BDD-based approach appears to be superior for IIS semantics, while the SAT-based approach appears to be superior for IS semantics.

The rest of the paper is organised as follows. In Sect. 2 we recall interpreted systems (IS), interleaved interpreted systems (IIS), the logic LTLK, and its two subsets: LTL and ELTLK (i.e., the existential fragment of LTLK). Section 3 deals with Bounded Model Checking (BMC), where Sect. 3.1 describes BDD-based BMC for ELTLK and Sect. 3.2 presents SAT-based BMC for ELTLK. In the last section we discuss our experimental results and conclude the paper.

1.1 Related work

Model checking of knowledge properties was first considered by Vardi and Halpern [20]. The complexity of the model checking problem for LTL combined with epistemic modalities

in the perfect recall semantics was studied by van der Meyden and Shilov [38]. Raimondi et al. showed a BDD-based method for model checking CTLK[43]. Su et al [46]. described a method for model checking LTLK formulae using BDDs. Hoek et al. [22] proposed a method for model checking LTLK formulae using the logic of local propositions.

The origins of bounded model checking (BMC) go back to the seminal papers [4] and [3], where the method has been defined for the LTL properties and Boolean circuits. The main motivation of defining BMC was to take advantage of the immense success of SAT-solvers (i.e., tools implementing algorithms solving the satisfiability problem for propositional formulas). The first SAT-based BMC method for MAS was proposed in [41]. It deals with the existential fragment of the branching time logic extended with the epistemic components (ECTLK) and the interpreted systems. An implementation and experimental evaluation of this BMC method for the interleaved interpreted systems have been presented in [29]. For the same logic and for the standard interpreted systems, Jones et al. proposed a BMC method based on BDDs [26]. In [53] the SAT-based BMC method for the existential fragment of RTCTL augmented to include epistemic modalities (RTECTLK) and for the interleaved interpreted systems was introduced and experimentally evaluated. This BMC encoding takes into account the substantial improvement of the BMC encoding for ECTL that has been defined in [54]. Further, since RTECTLK is an extension of ECTLK such that a range of every temporal operator can be bounded, the BMC encoding of [53] substantially improves the BMC encoding presented in [29,41]. In [37] a BDD-based BMC method for RTECTLK over interleaved interpreted systems was defined and compared to the corresponding SAT-based BMC method. Further, in [49] the SAT-based BMC method for the deontic interpreted systems and for ECTLK extended to include the existential deontic modalities was defined. A more efficient translation to SAT together with an implementation and an experimental evaluation of this BMC method are shown in [51], where the SAT-based BMC method for RTECTLK augmented to include the existential deontic modalities was defined. In [23] a new SAT-based BMC encoding for fair ECTLK was presented. Next, in [32] the SAT-based BMC method for the real-time interpreted systems and for the existential fragment of TCTL extend to include epistemic modalities was shown. All the above BMC approaches deal with the properties expressed in the existential fragments of branching time temporal logics only.

For the linear time temporal-epistemic properties, until now, the following BMC methods have been developed. In [42] a SAT-based BMC method for ELTLK over interleaved interpreted systems has been defined. The main difficulty in the extension of the SAT-based BMC method for ELTL to the properties expressible in ELTLK was in the encoding of the looping conditions. This difficulty arises from the fact that in SAT-based BMC for ELTLK we need to consider more than one path. The BMC encoding presented in [42] is not based on the state-of-the-art BMC method for ECTL* [55], which uses a reduced number of paths and a more efficient encoding of loops, what results in significantly smaller and less complicated propositional formulae that encode the ELTLK properties. For the same logic over the same systems, in [33] a BDD-based BMC method was introduced. Next, in [52] a SAT-based BMC method for the existential fragment of Metric LTL with epistemic and deontic modalities (EMTLKD) over deontic interleaved interpreted systems was defined.

The usefulness of SAT-based BMC for error tracking and complementarity to the BDD-based symbolic model checking have already been proven in several works, e.g., [9,13,35,36]. Further, in [34] the semantics of interpreted systems and interleaved interpreted systems were experimentally evaluated by means of the BDD-based BMC method for LTLK. Partial-order reductions for model checking of interleaved interpreted systems were presented in [31].

Table 1 Summary of the tools and model checking techniques for temporal-epistemic-deontic logics

	SAT-BMC	BDD-BMC	NOT BMC
CTLK	VerICS/IIS, MCK/IS	MCMAS/IS	MCMAS/IS, MCK/IS
CTLKD	VerICS/IIS		MCMAS/IS
LTLK	VerICS/IS+IIS	VerICS/IS+IIS	MCK/IS
CTL*K	MCK/IS		MCK/IS
RTCTLK	VerICS/IIS	VerICS/IIS	
RTCTLKD	VerICS/IIS		

The BMC methods are defined for the existential fragments of the mentioned logics

Table 2 Summary of the BMC techniques for temporal-epistemic-deontic logics

	SAT-BMC		BDD-BMC	
	IIS	IS	IIS	IS
ELTLK	[42]		[33]	[34]
ECTLK	[29]	[41][a], [23]	[26]	
ECTLKD		[49][a]		
RTECTLK	[53]		[37]	
RTECTLKD	[51]			
TECTLK	[32]			
EMTLKD	[52]			

[a] we denote the BMC methods that have not been implemented

Table 1 provides a summary of the existing implementations of model checking techniques for MAS in the BMC context. Table 2 summarises the existing BMC techniques for MAS.

This paper combines and refines our preliminary results published in informal proceedings of two workshops: the CS&P'2011 [33] and the LAM'2012 [34], in the conference paper [36], and in the journal [42]. More precisely, for the interleaved interpreted systems and for the ELTLK properties we present a BDD-based BMC technique and an improved SAT-based BMC method that previously appeared in, respectively, [33,36] and [36,42]. For the interpreted systems and for the ELTLK properties we present a BDD-based BMC technique that previously appeared in [34]. Both the SAT-based BMC method are based on the SAT-based BMC technique for ECTL* that was introduced in [55].

2 Preliminaries

In this section we introduce the basic definitions used in the paper. In particular, we define interpreted and interleaved interpreted systems, and syntax and semantics of linear temporal logic extended with the epistemic component (LTLK) and its two subsets ELTLK and LTL.

2.1 Interpreted systems

The semantics of interpreted systems (IS) provides a setting to reason about multi-agent systems (MASs) by means of specifications based on knowledge and linear or branching time. We report here the basic setting as popularised in [16].

We begin by assuming that a MAS is composed of n agents (by $\mathcal{A} = \{1, \ldots, n\}$ we denote the non-empty set of agents) and a special agent e which is used to model the environment in which the agents operate. We associate a set of *possible local states* L_c and *actions* Act_c to each agent $c \in \mathcal{A} \cup \{e\}$. For any agent $c \in \mathcal{A} \cup \{e\}$ we assume that the special action ϵ_c, called the "null" action of agent c, belongs to Act_c. For convenience, the symbol Act denotes the Cartesian product of the agents' actions, i.e. $Act = Act_1 \times \cdots \times Act_n \times Act_e$.

An element $a \in Act$ is a tuple of actions (one for each agent) and is referred to as a *joint action*. Following closely the interpreted system model, we consider a *local protocol* modelling the program the agent is executing. Formally, for any agent $c \in \mathcal{A} \cup \{e\}$, the actions of the agents are selected according to a *local protocol* function $P_c : L_c \to 2^{Act_c}$, which maps local states to sets of possible actions for agent c. Further, for each agent c we define a (partial) evolution function $t_c : L_c \times Act \to L_c$. We assume that if $\epsilon_c \in P_c(\ell)$, then $t_c(\ell, (a_1, \ldots, a_n, a_e)) = \ell$ for $a_c = \epsilon_c$ and $a_i \in Act_i$ for $1 \leqslant i \leqslant n$, and $a_e \in Act_e$.

A *global state* $g = (\ell_1, \ldots, \ell_n, \ell_e)$ 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 $g = (\ell_1, \ldots, \ell_n, \ell_e)$, we denote by $l_c(g) = \ell_c$ the local component of agent $c \in \mathcal{A} \cup \{e\}$ in g.

Let G be a set of global states. For a given set of agents $\mathcal{A}$, the environment e, and a set of propositional variables $\mathcal{PV}$, which can be either true or false, an *interpreted system* is a tuple

$$\text{IS} = (\iota, \{L_c, Act_c, P_c, t_c\}_{c \in \mathcal{A} \cup \{e\}}, \mathcal{V})$$

where $\iota \in G$ is the initial global state, and $\mathcal{V} : G \to 2^{\mathcal{PV}}$ is a valuation function.

Given the notions above we can now define formally the global (partial) evolution function. Namely, the *global (partial) evolution* function $t : G \times Act \to G$ is defined as follows: $t(g, a) = g'$ iff for all $c \in \mathcal{A}$, $t_c(l_c(g), a) = l_c(g')$ and $t_e(l_e(g), a) = l_e(g')$. In brief we write the above as $g \xrightarrow{a} g'$.

With each IS we associate a Kripke *model*, which is a tuple

$$M = (G, \iota, T, \{\sim_c\}_{c \in \mathcal{A}}, \mathcal{V})$$

where $G = \prod_{c=1}^{n} L_c \times L_e$ is a set of the global states, $\iota \in G$ is the initial (global) state, $T \subseteq G \times G$ is a global transition relation on G defined by: $(g, g') \in T$ iff there exists an action $a \in Act$ such that $g \xrightarrow{a} g'$. We assume that the relation is total, i.e., for any $g \in G$ there exists an $a \in Act$ such that $g \xrightarrow{a} g'$ for some $g' \in G$, $\sim_c \subseteq G \times G$ is an *epistemic indistinguishability* relation for each agent $c \in \mathcal{A}$, defined by $g \sim_c r$ if $l_c(g) = l_c(r)$, and $\mathcal{V} : G \to 2^{\mathcal{PV}}$ is the valuation function of IS.

2.2 Interleaved interpreted systems

Interleaved interpreted systems (IIS) [31] are a restriction of interpreted systems, where all the joint actions are of special form. To be more precise, we assume that if more than one agent is active at a given state, i.e., executes a non null-action, then all the active agents perform the same (shared) action in the round. Formally, for any agent $c \in \mathcal{A} \cup \{e\}$ we assume that the special action ϵ_c, called "null" action of agent c, belongs to Act_c; as it will become clear below the local state of agent c remains the same if the null action is performed. Next, $Act = \bigcup_{c \in \mathcal{A}} Act_c \cup Act_e$, and for each action a, by $Agent(a) \subseteq \mathcal{A} \cup \{e\}$

we mean all the agents c such that $a \in Act_c$, i.e., the set of agents potentially able to perform a. Further, for each agent $c \in \mathcal{A} \cup \{e\}$, the actions are selected according to a *local protocol* function $P_c : L_c \to 2^{Act_c}$ such that $\epsilon_c \in P_c(\ell)$, for any $\ell \in L_c$, i.e., we insist on the null action to be enabled at every local state. Next, for each agent $c \in \mathcal{A} \cup \{e\}$, we define a (partial) evolution function $t_c : L_c \times Act_c \to L_c$, where $t_c(\ell, \epsilon_c) = \ell$ for each $\ell \in L_c$. The local evolution function considered here differs from the standard treatment in interpreted systems by having the local action as the parameter instead of the joint action.

Let G be a set of global states. For a given set of agents $\mathcal{A}$, the environment e, and a set of propositional variables $\mathcal{PV}$, which can be either true or false, an *interleaved interpreted system* is a tuple

$$\text{IIS} = (\iota, \{L_c, Act_c, P_c, t_c\}_{c \in \mathcal{A} \cup \{e\}}, \mathcal{V})$$

where $\iota \in G$ is the initial global state, and $\mathcal{V} : G \to 2^{\mathcal{PV}}$ is a valuation function.

Given the notions above we can now define formally the global (partial) interleaved evolution function. Namely, the *global (partial) interleaved evolution* function $t : G \times \prod_{c=1}^{n} Act_c \times Act_e \to G$ is defined as follows: $t(g, a_1, \ldots, a_n, a_e) = g'$ iff there exists an action $a \in Act \setminus \{\epsilon_1, \ldots, \epsilon_n, \epsilon_e\}$ such that for all $c \in Agent(a)$, $a_c = a$ and $t_c(l_c(g), a) = l_c(g')$, and for all $c \in (\mathcal{A} \cup \{e\}) \setminus Agent(a)$, $a_c = \epsilon_c$ and $t_c(l_c(g), \epsilon_c) = l_c(g)$. In brief we write the above as $g \xrightarrow{a} g'$.

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 that the local protocols are defined to permit this; if a local protocol does not allow it, then the local action cannot be performed and therefore the global transition does not comply with the global interleaved evolution function as defined above.

With each IIS we associate a Kripke *model*, which is a tuple

$$M = (G, \iota, T, \{\sim_c\}_{c \in \mathcal{A}}, \mathcal{V})$$

where $G = \prod_{c=1}^{n} L_c \times L_e$ is a set of the global states, $\iota \in G$ is the initial (global) state, $T \subseteq G \times G$ is a global (interleaved) transition relation on G defined by: $(g, g') \in T$ iff there exists an action $a \in Act \setminus \{\epsilon_1, \ldots, \epsilon_n, \epsilon_e\}$ such that $g \xrightarrow{a} g'$. We assume that the relation is total, i.e., for any $g \in G$ there exists an $a \in Act \setminus \{\epsilon_1, \ldots, \epsilon_n, \epsilon_e\}$ such that $g \xrightarrow{a} g'$ for some $g' \in G$, $\sim_c \subseteq G \times G$ is an *epistemic indistinguishability* relation for each agent $c \in \mathcal{A}$, defined by $g \sim_c r$ if $l_c(g) = l_c(r)$, and $\mathcal{V} : G \to 2^{\mathcal{PV}}$ is the valuation function of IIS.

2.3 Runs and paths

Let M be a model generated by either IS or IIS. Then, an infinite sequence of global states $\rho = g_0 g_1 g_2 \ldots$ is called a *run* originating at g_0 if there is a sequence of transitions from g_0 onwards, such that, $(g_i, g_{i+1}) \in T$ for every $i \geqslant 0$. The m-th prefix of ρ, denoted by $\rho[..m]$, is defined as $\rho[..m] = (g_0, g_1, \ldots, g_m)$. Any finite prefix of a run is called a *path*.

By $length(\rho)$ we mean the number of the states of ρ if ρ is a path, and ω if ρ is a run. In order to limit the indices range of ρ, which can be either a path or a run, we define the relation $\trianglelefteq_\rho$. Let $\trianglelefteq_\rho \stackrel{def}{=} <$ if ρ is a run, and $\trianglelefteq_\rho \stackrel{def}{=} \leqslant$ if ρ is a path.

The set of all the paths and runs originating from g is denoted by $\Pi(g)$. The set of all the paths and runs originating from all states in G is defined as $\Pi = \bigcup_{g \in G} \Pi(g)$. The set of all

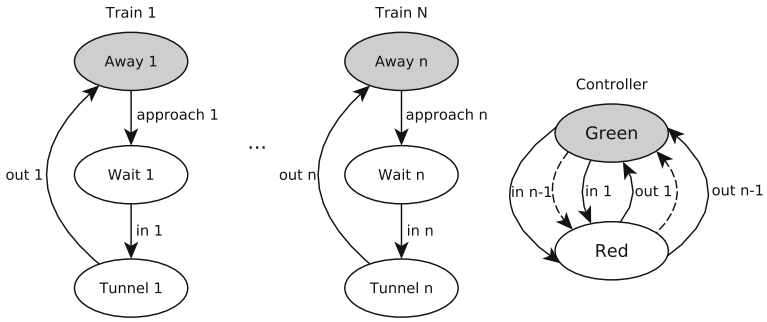


Fig. 1 The FTC system

the runs originating from g is denoted by $\Pi^{\omega}(g)$. The set of all the runs originating from all states in G is defined as $\Pi^{\omega} = \bigcup_{g \in G} \Pi^{\omega}(g)$. A state g is *reachable* from g_0 if there is a path $\rho = g_0 g_1 g_2 \ldots g_n$ for $n \geqslant 0$ such that $g = g_n$.

2.4 Examples of MASs and their models

In the section we present MASs modelled by means of interpreted systems and interleaved interpreted systems. We use the systems to appraise the bounded model checking methods considered in the paper. In what follows we denote by $\overline{\epsilon}$ the joint null action, i.e., the action composed of the null actions only.

2.4.1 A faulty train controller system (FTC)

The FTC (adapted from [21]) consists of a controller, and n trains (for $n \geqslant 2$), one of which is dysfunctional. It is assumed that each train uses its own circular track for travelling in one direction. At one point, all trains have to pass through a tunnel, but because there is only one track in the tunnel, trains arriving from each direction cannot use it simultaneously. There are signals on both sides of the tunnel, which can be either red or green. All trains except one with a faulty signalling system notify the controller when they request entry to the tunnel or when they leave the tunnel. The controller controls the colour of the displayed signal. Figure 1 shows the local states, the possible actions, and the protocol for each agent. Null actions are omitted in the figure. Further, we assume that the local state $Away_i$ is initial for Train i, and the local state $Green$ is initial for Controller.

In the model we assume the following set of proposition variables: $\mathcal{PV} = \{InTunnel_1, \ldots, InTunnel_n\}$ with the following interpretation: $(M, g) \models InTunnel_i$ if $l_{Train_i}(g) = Tunnel_i$ i for all $i \in \{1, \ldots, n\}$.

Let $state$ denote a local state of an agent, $Act = Act_{Train_1} \times \cdots \times Act_{Train_n} \times Act_{Controller}$ with $Act_{Train_i} = \{approach_i, in_i, out_i, \epsilon_i\}$ where $1 \leqslant i \leqslant n$, and $Act_{Controller} = \bigcup_{i=1}^{n-1} \{in_i, out_i\} \cup \{\epsilon\}$. Moreover, let $a \in Act$, $act_i(a)$ denote an action of Train i, and $act_C(a)$ denote an action of Controller. In the IS model of the system we assume the following local evolution functions:

- Let $1 \leqslant i \leqslant n$. The local evolution function for Train i is defined as follows:
  - $t_{Train_i}(state, a) = state$ if $a \neq \overline{\epsilon}$ and $act_i(a) = \epsilon_i$
  - $t_{Train_i}(Away_i, a) = Wait_i$ if $act_i(a) = approach_i$

- $t_{Train_i}(Wait_i, a) = Tunnel_i$ if $act_i(a) = in_i$ and $act_C(a) = in_i$ and $i \neq n$
- $t_{Train_i}(Tunnel_i, a) = Away_i$ if $act_i(a) = out_i$ and $act_C(a) = out_i$ and $i \neq n$
- $t_{Train_n}(Wait_n, a) = Tunnel_n$ if $act_n(a) = in_n$
- $t_{Train_n}(Tunnel_n, a) = Away_n$ if $act_n(a) = out_n$

– the local evolution function for Controller is defined as follows:

- $t_{Controller}(state, a) = state$ if $act_C(a) = \epsilon$
- $t_{Controller}(Green, a) = Red$ if $act_i(a) = in_i$ and $act_C(a) = in_i$ and $i \neq n$
- $t_{Controller}(Red, a) = Green\ act_i(a) = out_i$ and $act_C(a) = out_i$ and $i \neq n$

In the IIS model of the system we assume the following local evolution functions:

– for Train i, t_{Train_i} is defined as follows:

- $t_{Train_i}(state, \epsilon_i) = state$, for $1 \leqslant i \leqslant n$
- $t_{Train_i}(Away_i, approach_i) = Wait_i$, for $1 \leqslant i \leqslant n$
- $t_{Train_i}(Wait_n, in_n) = Tunnel_n$
- $t_{Train_i}(Wait_i, in_i) = Tunnel_i$ if $act_C(a) = in_i$ and $act_j(a) = \epsilon_j$ for all $1 \leqslant j < n$ such that $j \neq i$
- $t_{Train_n}(Tunnel_n, out_n) = Away_n$
- $t_{Train_i}(Tunnel_i, out_i) = Away_i$ if $act_C(a) = out_i$ and $act_j(a) = \epsilon_j$ for all $1 \leqslant j < n$ such that $j \neq i$

– for Controller, $t_{Controller}$ is defined as follows:

- $t_{Controller}(state, \epsilon) = state$
- $t_{Controller}(Green, in_i) = Red$ if $act_i(a) = in_i$, for $1 \leqslant i < n$
- $t_{Controller}(Red, out_i) = Green$ if $act_i(a) = out_i$, for $1 \leqslant i < n$

2.4.2 Faulty generic pipeline paradigm (FGPP)

The FGPP (adapted from [40]) consists of the following agents: the Producer that is able to produce data, the Consumer that is able to receive data, a chain of n intermediate Nodes that are able to receive, process, and send data, and a chain of n Alarms that are enabled when some error occurs, i.e. the $Hung\text{-}upi$ ($1 \leqslant i \leqslant n$) operation is performed three times. If the $Hung\text{-}upi$ action is performed only once or only twice, than the system recovers from the error. Figure 2 shows the local states, the possible actions, and the protocol for each agent. From Fig. 2 we can also deduce the local evolution function of IIS. Null actions are omitted in the figure. Further, we assume that the following local states $ProdReady$, $NodeiReady$, $ConsReady$ and $AlarmiReady$ are initial, respectively, for Producer, Node i, Consumer, and Alarm i.

In the model we assume the following set of proposition variables: $\mathcal{PV} = \{ProdSend, ConsReady, Problem_1, \ldots, Problem_n, Repair_1, \ldots, Repair_n, Alarm_1Send, \ldots, Alarm_nSend\}$ with the following interpretation:

- $(M, g) \models ProdSend$ if $l_{Producer}(g) = ProdSend$
- $(M, g) \models ConsReady$ if $l_{Consumer}(g) = ConsReady$
- $(M, g) \models Problem_i$ if $l_{Alarmi}(g) = Problemi$, for all $1 \leqslant i \leqslant n$
- $(M, g) \models Repair_i$ if $l_{Alarmi}(g) = Repairi$, for all $1 \leqslant i \leqslant n$
- $(M, g) \models Alarm_iSend$ if $l_{Alarmi}(g) = AlarmiSend$, for all $1 \leqslant i \leqslant n$

Let $state$ denote a local state of an agent, P, C, Ni, and Ai denote, respectively, Producer, Consumer, the i-th Node, and the i-th Alarm. Further, let $Act = Act_P \times$

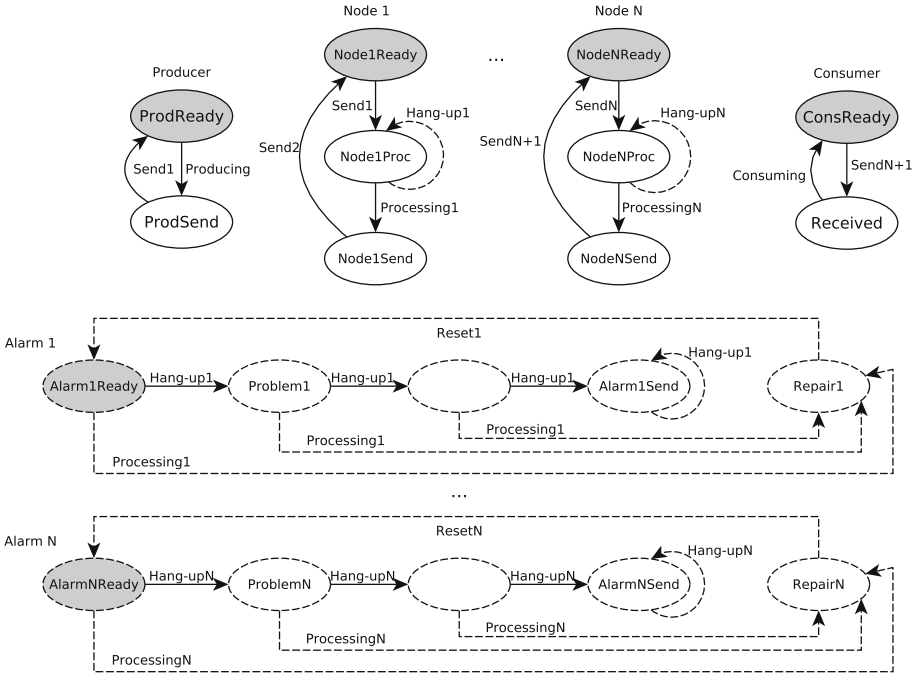


Fig. 2 The FGPP system. *Dashed lines* correspond to the system behaviour after an error has occured

$\prod_{i=1}^{n} Act_{Ni} \times \prod_{i=1}^{n} Act_{Ai} \times Act_C$ with $Act_P = \{Producing, Send_1, \epsilon_P\}$, $Act_C = \{Send_{n+1}, Consuming, \epsilon_C\}$, $Act_{Ni} = \{Send_i, Send_{i+1}, Processing_i, Hang\_up_i, \epsilon_{Ni}\}$, and $Act_{Ai} = \{Processing_i, Hang\_up_i, Reset_i, \epsilon_{Ai}\}$. Moreover, let $a \in Act$, and $act_P(a)$, $act_{Ni}(a)$, $act_{Ai}(a)$, and $act_C(a)$, respectively, denote an action of Producer, Node i, Alarm i, and Consumer. In the IS model of the system we assume the following local evaluation functions:

- $t_P(state, a) = state$ if $a \neq \overline{\epsilon}$ and $act_P(a) = \epsilon_P$
- $t_P(ProdReady, a) = ProdSend$ if $act_P(a) = Producing$
- $t_P(ProdSend, a) = ProdReady$ if $act_P(a) = Send_1$ and $act_{N1}(a) = Send_1$
- $t_C(state, a) = state$ if $act_C(a) = \epsilon_C$
- $t_C(ConsReady, a) = Received$ if $act_C(a) = Send_{n+1}$ and $act_{Nn}(a) = Send_{n+1}$
- $t_C(Received, a) = ConsReady$ if $act_C(a) = Consuming$
- if $n = 1$
  - $t_{N1}(state, a) = state$ if $a \neq \overline{\epsilon}$ and $act_{N1}(a) = \epsilon_{N1}$
  - $t_{N1}(Node1Ready, a) = Node1Proc$ if $act_{N1}(a) = act_P(a) = Send_1$
  - $t_{N1}(Node1Proc, a) = Node1Send$ if $act_{N1}(a) = act_{A1}(a) = Processing_1$
  - $t_{N1}(Node1Proc, a) = Node1Proc$ if $act_{N1}(a) = act_{A1}(a) = Hang\_up_1$
  - $t_{N1}(Node1Send, a) = Node1Ready$ if $act_{N1}(a) = act_C(a) = Send_2$
- if $n = 2$
  - $t_{N1}(state, a) = state$ if $a \neq \overline{\epsilon}$ and $act_{N1}(a) = \epsilon_{N1}$
  - $t_{N1}(Node1Ready, a) = Node1Proc$ if $act_{N1}(a) = act_P(a) = Send_1$
  - $t_{N1}(Node1Proc, a) = Node1Send$ if $act_{N1}(a) = act_{A1}(a) = Processing_1$

- $t_{N1}(Node1Proc, a) = Node1Proc$ if $act_{N1}(a) = act_{A1}(a) = Hang\_up_1$
- $t_{N1}(Node1Send, a) = Node1Ready$ if $act_{N1}(a) = act_{N2}(a) = Send_2$
- $t_{N2}(state, a) = state$ if $a \neq \overline{\epsilon}$ and $act_{N2}(a) = \epsilon_{N2}$
- $t_{N2}(Node2Ready, a) = Node2Proc$ if $act_{N2}(a) = act_{N1}(a) = Send_2$
- $t_{N2}(Node2Proc, a) = Node2Send$ if $act_{N2}(a) = act_{A2}(a) = Processing_2$
- $t_{N2}(Node2Proc, a) = Node2Proc$ if $act_{N2}(a) = act_{A2}(a) = Hang\_up_2$
- $t_{N2}(Node2Send, a) = Node2Ready$ if $act_{N2}(a) = act_C(a) = Send_3$

– if $n \geqslant 3$ and $2 \leqslant i < n$

- $t_{N1}(state, a) = state$ if $a \neq \overline{\epsilon}$ and $act_{N1}(a) = \epsilon_{N1}$
- $t_{N1}(Node1Ready, a) = Node1Proc$ if $act_{N1}(a) = act_P(a) = Send_1$
- $t_{N1}(Node1Proc, a) = Node1Send$ if $act_{N1}(a) = act_{A1}(a) = Processing_1$
- $t_{N1}(Node1Proc, a) = Node1Proc$ if $act_{N1}(a) = act_{A1}(a) = Hang\_up_1$
- $t_{N1}(Node1Send, a) = Node1Ready$ if $act_{N1}(a) = act_{N2}(a) = Send_2$
- $t_{Nn}(state, a) = state$ if $a \neq \overline{\epsilon}$ and $act_{Nn}(a) = \epsilon_{Nn}$
- $t_{Nn}(NodeNReady, a) = NodeNProc$ if $act_{Nn}(a) = act_{Nn-1}(a) = Send_n$
- $t_{Nn}(NodeNProc, a) = NodeNSend$ if $act_{Nn}(a) = act_{An}(a) = Processing_n$
- $t_{Nn}(NodeNProc, a) = NodeNProc$ if $act_{Nn}(a) = act_{An}(a) = Hang\_up_n$
- $t_{Nn}(NodeNSend, a) = NodeNReady$ if $act_{Nn}(a) = act_C(a) = Send_{n+1}$
- $t_{Ni}(state, a) = state$ if $a \neq \overline{\epsilon}$ and $act_{Ni}(a) = \epsilon_{Ni}$
- $t_{Ni}(NodeNReady, a) = NodeNProc$ if $act_{Ni}(a) = act_{Nn-1}(a) = Send_i$
- $t_{Ni}(NodeNProc, a) = NodeNSend$ if $act_{Ni}(a) = act_{Ai}(a) = Processing_i$
- $t_{Ni}(NodeNProc, a) = NodeNProc$ if $act_{Ni}(a) = act_{Ai}(a) = Hang\_up_i$
- $t_{Ni}(NodeNSend, a) = NodeNReady$ if $act_{Ni}(a) = act_{Ni+1}(a) = Send_{i+1}$

– Let $1 \leqslant i \leqslant n$:

- $t_{Ai}(state, a) = state$ if $a \neq \overline{\epsilon}$ and $act_{Ai}(a) = \epsilon_{Ai}$
- $t_{Ai}(AlarmiReady, a) = Problemi$ if $act_{Ai}(a) = act_{Ni}(a) = Hang\_up_i$
- $t_{Ai}(AlarmiReady, a) = Repairi$ if $act_{Ai}(a) = act_{Ni}(a) = Processing_i$
- $t_{Ai}(Problemi, a) = Problemi'$ if $act_{Ai}(a) = act_{Ni}(a) = Hang\_up_i$
- $t_{Ai}(Problemi, a) = Repairi$ if $act_{Ai}(a) = act_{Ni}(a) = Processing_i$
- $t_{Ai}(Problemi', a) = AlarmiSend$ if $act_{Ai}(a) = act_{Ni}(a) = Hang\_up_i$
- $t_{Ai}(Problemi', a) = Repairi$ if $act_{Ai}(a) = act_{Ni}(a) = Processing_i$
- $t_{Ai}(AlarmiSend, a) = AlarmiSend$ if $act_{Ai}(a) = act_{Ni}(a) = Hang\_up_i$
- $t_{Ai}(Repairi, a) = AlarmiReady$ if $act_{Ai}(a) = Reseti$.

2.4.3 Dining cryptographers (DC)

The DC [10] is a scalable anonymity protocol, which has been formalised and analysed in many works, e.g., [27,39]. Our formalisation of DC is shown in Fig. 3 and extends our earlier definition [27]. Null actions are omitted in the figure.

We model n cryptographers sitting at a round table, with coins between them, every coin seen by a pair of respective neighbours. Let $state$ denote a local state of an agent. Let C_i and $Coin_i$ denote the i-th cryptographer and i-th coin, respectively. $Counter$ denotes the agent counting utterances and $Oracle_i$ determines if the agent i pays, or no agent pays at all. Thus, our DC system consists of $3n + 1$ components formed by n agents and the environment. More precisely, the i-th agent consists of the following three components: C_i, $Coin_i$, and $Oracle_i$. The component $Counter$ defines the environment. We introduce a helper function

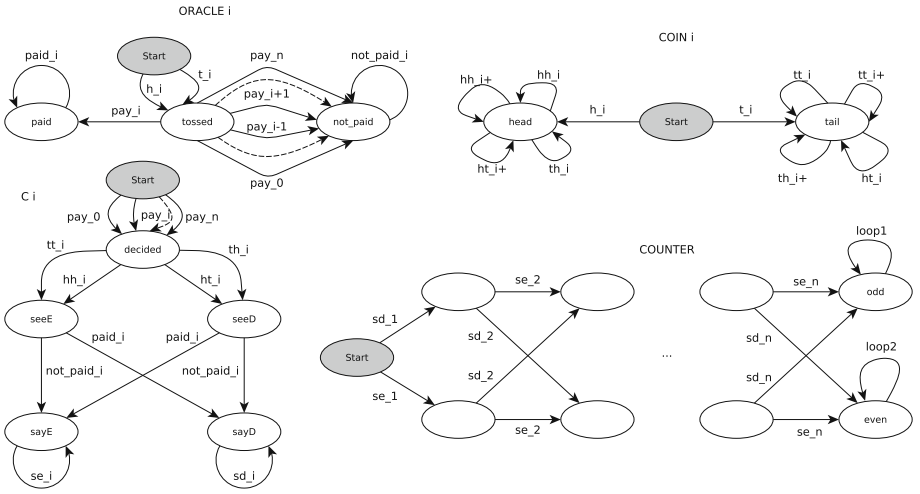


Fig. 3 Dining cryptographers (DC)

to identify the right-side neighbour of the cryptographer i: $i^+ = (i + 1)$ for $1 \leqslant i < n$, and $i^+ = 1$ for $i = n$.

The protocol works as follows: first the oracles determine who is the payer (either precisely one cryptographer or none of them). Then, every cryptographer looks at the two coins he can see (his and his right neighbour), and records the result (the states $seeD$ and $seeE$ correspond to seeing either different or equal coin sides, respectively). The final utterance of each cryptographer ($sayD$ and $sayE$ locations correspond to saying different and equal outcomes, respectively) depends of what result is seen and whether the cryptographer has paid or not. Finally, the counter counts the utterances, determining the final result of the protocol. Let $Act = Act_{Counter} \times \prod_{i=1}^{n} Act_{C_i} \times \prod_{i=1}^{n} Act_{Coin_i} \times \prod_{i=1}^{n} Act_{Oracle_i}$ with

- $Act_{Counter} = \{se_1, sd_1, \cdots, se_n, sd_n, \epsilon_{Counter}\}$,
- $Act_{Coin_i} = \{tt_i, hh_i, ht_i, th_i, tt_{i^+}, hh_{i^+}, ht_{i^+}, th_{i^+}, \epsilon_{Coin_i}\}$,
- $Act_{Oracle_i} = \{pay_0, \ldots, pay_n, t_i, h_i paid_i, not\_paid_i, \epsilon_{Oracle_i}\}$, and
- $Act_{C_i} = \{pay_0, \ldots, pay_n, tt_i, hh_i, ht_i, th_i, not\_paid_i, paid_i, se_i, sd_i, \epsilon_{C_i}\}$,

for all $1 \leqslant i \leqslant n$. Moreover, let $a \in Act$, and $act_{Counter}(a)$, $act_{C_i}(a)$, $act_{Coin_i}(a)$, and $act_{Oracle}(a)$, respectively, denote an action of Oracle, Cryptographer i, Coin i, and Counter.

In the IS model of the system we assume the following local evolution functions (we provide definitions for C_i and $Oracle_i$ components, the remaining ones are straightforward):

- the local evolution for $Oracle_i$ is defined as follows:
  - $t_{Oracle_i}(state, a) = state$ iff $a \neq \overline{\epsilon}$ and $act_{Oracle_i}(a) = \epsilon_{Oracle_i}$
  - $t_{Oracle_i}(start, a) = tossed$ iff $act_{Oracle_i}(a) = act_{Coin_i}(a) = t_i$ or $act_{Oracle_i}(a) = act_{Coin_i}(a) = h_i$
  - $t_{Oracle_i}(tossed, a) = paid$ iff $act_{Oracle_1}(a) = \ldots = act_{Oracle_n}(a) = pay_i$ and $act_{C_1}(a) = \ldots = act_{C_n}(a) = pay_i$
  - $t_{Oracle_i}(tossed, a) = not\_paid$ iff either $act_{Oracle_1}(a) = \ldots = act_{Oracle_n}(a) = pay_0$ and $act_{C_1}(a) = \ldots = act_{C_n}(a) = pay_0$, or $act_{Oracle_1}(a) = \ldots = act_{Oracle_n}(a) = pay_j$ and $act_{C_1}(a) = \ldots = act_{C_n}(a) = pay_j$ for some j such that $1 \leqslant j \leqslant n$ and $j \neq i$

– the local evolution for C_i is defined as follows:
  - $t_{C_i}(state, a) = state$ iff $a \neq \overline{\epsilon}$ and $act_{C_i}(a) = \epsilon_{C_i}$
  - $t_{C_i}(start, a) = decided$ iff $act_{Oracle_1}(a) = \ldots = act_{Oracle_n}(a) = pay_j$ and $act_{C_1}(a) = \ldots = act_{C_n}(a) = pay_j$ for some j such that $0 \leqslant j \leqslant n$
  - $t_{C_i}(decided, a) = seeD$ iff $act_{C_i}(a) = act_{Coin_i}(a) = act_{Coin_{i+}}(a) = th_i$
  - $t_{C_i}(decided, a) = seeD$ iff $act_{C_i}(a) = act_{Coin_i}(a) = act_{Coin_{i+}}(a) = ht_i$
  - $t_{C_i}(decided, a) = seeE$ iff $act_{C_i}(a) = act_{Coin_i}(a) = act_{Coin_{i+}}(a) = hh_i$
  - $t_{C_i}(decided, a) = seeE$ iff $act_{C_i}(a) = act_{Coin_i}(a) = act_{Coin_{i+}}(a) = tt_i$
  - $t_{C_i}(seeE, a) = sayD$ iff $act_{C_i}(a) = act_{Oracle_i}(a) = paid_i$
  - $t_{C_i}(seeD, a) = sayE$ iff $act_{C_i}(a) = act_{Oracle_i}(a) = paid_i$
  - $t_{C_i}(seeD, a) = sayD$ iff $act_{C_i}(a) = act_{Oracle_i}(a) = not\_paid_i$
  - $t_{C_i}(seeE, a) = sayE$ iff $act_{C_i}(a) = act_{Oracle_i}(a) = not\_paid_i$

Because of the way in which the local evolution functions are defined obtaining the global evolution function for IIS requires only that the components not mentioned in every of the above definitions, execute their respective ϵ actions. For example, because we provide separate actions for every payment configuration, there is no need to enforce any additional conditions at the global level.

In the model we assume the following set of propositional variables: $\mathcal{PV} = \{odd, paid_1, \ldots paid_n\}$ with the following interpretation:

– $(M, g) \models odd$ if $l_{Counter}(g) = odd$,
– $(M, g) \models paid_i$ if $l_{Oracle_i}(g) = paid$, for all $1 \leqslant i \leqslant n$.

2.5 LTLKand its two subsets: ELTLKand LTL

Combinations of linear time with knowledge have long been used in the analysis of temporal epistemic properties of multi-agent systems [16]. We now recall the basic definitions and adapt them to our purposes when needed.

2.5.1 Syntax

Let $\mathcal{PV}$ be a set of propositional variables to be interpreted over the global states of a system, $p \in \mathcal{PV}$, and $\Gamma \subseteq \mathcal{A}$. The LTLK formulae in the negation normal form are given by the following grammar:

$$\varphi ::= true \mid false \mid p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathrm{X}\varphi \mid \varphi \mathrm{U} \varphi \mid \varphi \mathrm{R} \varphi \mid \mathrm{K}_c\varphi \mid \overline{\mathrm{K}}_c\varphi \mid \mathrm{E}_\Gamma\varphi \mid \overline{\mathrm{E}}_\Gamma\varphi \mid \mathrm{D}_\Gamma\varphi \mid \overline{\mathrm{D}}_\Gamma\varphi \mid \mathrm{C}_\Gamma\varphi \mid \overline{\mathrm{C}}_\Gamma\varphi.$$

The temporal modalities U and R are named as usual *until* and *release*, respectively, X is the next step modality. The derived basic temporal modalities are defined as follows: $\mathrm{F}\varphi \stackrel{def}{=} true \mathrm{U}\varphi$ and $\mathrm{G}\varphi \stackrel{def}{=} false \mathrm{R}\varphi$.

The epistemic operator $K_c\varphi$ represents "agent c knows φ" while the operator $\overline{\mathrm{K}}_c\varphi$ is the corresponding dual one representing "agent c considers φ possible". The epistemic operators D_Γ, E_Γ, and C_Γ represent distributed knowledge in the group Γ, "everyone in Γ knows", and common knowledge among agents in Γ, respectively. The epistemic operator $\overline{\mathrm{D}}_\Gamma$, $\overline{\mathrm{E}}_\Gamma$, and $\overline{\mathrm{C}}_\Gamma$ are the corresponding dual ones.

Note that LTL is the sublogic of LTLK which consists only of the formulae built without the epistemic operators, i.e., LTL formulae are defined by the following grammar:

$$\varphi ::= true \mid false \mid p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathrm{X}\varphi \mid \varphi \mathrm{U} \varphi \mid \varphi \mathrm{R} \varphi.$$

ELTLKis the existential fragment of LTLK, defined by the following grammar:

$$\varphi ::= true \mid false \mid p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathrm{X}\varphi \mid \varphi \mathrm{U} \varphi \mid \varphi \mathrm{R} \varphi \mid \overline{\mathrm{K}}_c \varphi \mid \overline{\mathrm{E}}_\Gamma \varphi \mid \overline{\mathrm{D}}_\Gamma \varphi \mid \overline{\mathrm{C}}_\Gamma \varphi.$$

Observe that we assume that the LTLK (and so LTL and ELTLK) formulae are given in the negation normal form (NNF), in which the negation can be only applied to propositional variables.

2.5.2 Semantics

Let $M = (G, \iota, T, \{\sim_c\}_{c \in \mathcal{A}}, \mathcal{V})$ be a model, and ρ be a path or run. By $\rho(i)$ we denote the i-th state of ρ, and by $\rho[m]$ we denote the path or run ρ with a designated formula evaluation position m, where $m \trianglelefteq_\rho length(\rho)$. Further, let $\Gamma \subseteq \mathcal{A}$. We use the following standard relations to give semantics to the "everyone knows", "common knowledge", and "distributed knowledge" modalities: $\sim^E_\Gamma = \bigcup_{c \in \Gamma} \sim_c$, $\sim^C_\Gamma$ is the transitive closure of $\sim^E_\Gamma$, whereas $\sim^D_\Gamma = \bigcap_{c \in \Gamma} \sim_c$.

We say that an LTLK formula φ is true along ρ (in symbols $M, \rho \models \varphi$) iff $M, \rho[0] \models \varphi$, where

$M, \rho[m] \models true$
$M, \rho[m] \not\models false$
$M, \rho[m] \models p$ iff $p \in \mathcal{V}(\rho(m))$
$M, \rho[m] \models \neg p$ iff $p \notin \mathcal{V}(\rho(m))$
$M, \rho[m] \models \varphi \wedge \psi$ iff $M, \rho[m] \models \varphi$ and $M, \rho[m] \models \psi$
$M, \rho[m] \models \varphi \vee \psi$ iff $M, \rho[m] \models \varphi$ or $M, \rho[m] \models \psi$
$M, \rho[m] \models \mathrm{X}\varphi$ iff $length(\rho) > m$ and $M, \rho[m+1] \models \varphi$
$M, \rho[m] \models \varphi \mathrm{U} \psi$ iff $(\exists k \geq m)(M, \rho[k] \models \psi$ and $(\forall m \leq j < k) M, \rho[j] \models \varphi)$
$M, \rho[m] \models \varphi \mathrm{R} \psi$ iff $(\rho \in \Pi^\omega(\iota)$ and $(\forall k \geq m) M, \rho[k] \models \psi)$ or
$(\exists k \geq m)(M, \rho[k] \models \varphi$ and $(\forall m \leq j \leq k) M, \rho[j] \models \psi)$
$M, \rho[m] \models \mathrm{K}_c \varphi$ iff $(\forall \rho' \in \Pi^\omega(\iota))(\forall k \geqslant 0)(\rho'(k) \sim_c \rho(m)$ implies $M, \rho'[k] \models \varphi)$
$M, \rho[m] \models \overline{\mathrm{K}}_c \varphi$ iff $(\exists \rho' \in \Pi(\iota))(\exists k \geqslant 0)(\rho'(k) \sim_c \rho(m)$ and $M, \rho'[k] \models \varphi)$
$M, \rho[m] \models \mathrm{Y}_\Gamma \varphi$ iff $(\forall \rho' \in \Pi^\omega(\iota))(\forall k \geqslant 0)(\rho'(k) \sim^{\mathrm{Y}}_\Gamma \rho(m)$ implies $M, \rho'[k] \models \varphi)$
$M, \rho[m] \models \overline{\mathrm{Y}}_\Gamma \varphi$ iff $(\exists \rho' \in \Pi(\iota))(\exists k \geqslant 0)(\rho'(k) \sim^{\mathrm{Y}}_\Gamma \rho(m)$ and $M, \rho'[k] \models \varphi)$,
where $\mathrm{Y} \in \{\mathrm{D}, \mathrm{E}, \mathrm{C}\}$.

Let g be a global state of M and φ an LTLK formula. We assume the following notations:

- $M, g \models \varphi$ iff $M, \rho \models \varphi$ for all the runs $\rho \in \Pi^\omega(g)$.
- $M \models \varphi$ iff $M, \iota \models \varphi$.
- $M, g \models^\exists \varphi$ iff $M, \rho \models \varphi$ for some path or run $\rho \in \Pi(g)$.
- $Props(\varphi)$ is the set of the propositional variables appearing in φ.

Let m be a formula evaluation position, and $p, q \in \mathcal{PV}$. An illustration of the semantics is shown in Figs. 4, 5, 6.
Given the above, we say that:

- the LTLK formula φ holds in the model M (written $M \models \varphi$) iff $M, \rho \models \varphi$ for all runs $\rho \in \Pi^\omega(\iota)$.

Fig. 4 Evaluation of formulae of types: Next state and Until

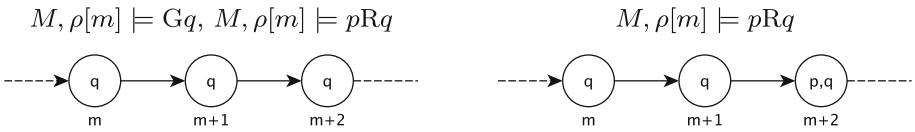


Fig. 5 Evaluation of formulae of the Relase type

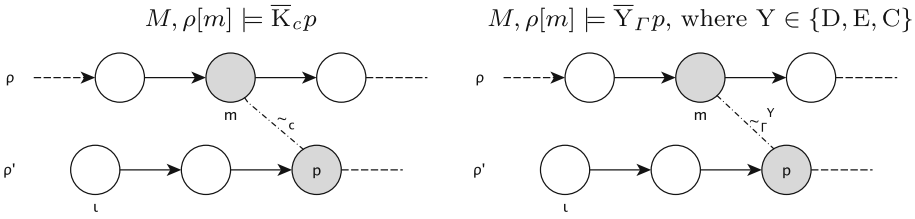


Fig. 6 Evaluation of existential epistemic formulae. The highlighted states are epistemically equivalent

– the ELTLK formula φ holds in the model M (written $M \models^{\exists} \varphi$) iff $M, \rho \models \varphi$ for some path or run $\rho \in \Pi(\iota)$.

Determining whether an LTLK formula φ is *existentially* (resp. *universally*) valid in a model M is called an *existential* (resp. *universal*) model checking problem. In other words, the *universal model checking problem* asks whether $M \models \varphi$ and the *existential model checking problem* asks whether $M \models^{\exists} \varphi$.

In order to solve the universal model checking problem, one can negate the formula and show that the existential model checking problem for the negated formula has no solution. Intuitively, we are trying to find a counterexample, and if we do not succeed, then the formula is universally valid. Now, since bounded model checking is designed for finding a solution to an existential model checking problem, in the paper we only consider the properties expressible in ELTLK. This is because finding a counterexample, for example, to $M \models \mathrm{GK}_c p$ corresponds to the question whether there exists a witness to $M \models^{\exists} \mathrm{F}\overline{\mathrm{K}}_c \neg p$.

Our semantics meets two important properties. Firstly, for LTLK the definition of validity in a model M uses runs only. Secondly, if we replace each Π with Π^{ω}, the semantics does not change as our models have total transition relations (each path is a prefix of some run). The semantics applied to submodels of M does not have the above property, but it preserves ELTLK over M, which is shown in Lemma 1. Moreover, note that in the above semantics while we define the until operator, ρ could be an arbitrary path or run (i.e., $\rho \in \Pi$). However, while we define the release operator, we insist on ρ to be a run that starts in the initial state on the part of the definition that corresponds to the globally operator.

2.6 Comments on IS and IIS

There are variety of models of multi-agent systems. A fundamental dimension along which this models differ is the degree to which the activity of agents is synchronised. At one

end of the spectrum is the *synchronous model* in which acting of agents proceeds in a sequence of rounds. In each round, an agent performs an action that affects the other agents, is affected by actions executed by the other agents in that round, and changes his/her state. All agents perform actions at exactly the same time. At the other end is the *asynchronous model* in which there is no bound on the amount of time that can elapse between agents' actions, and there is no bound on the time it can take for an agent to act. Between these extremes there are the semi-synchronous models in which times of agents' actions can vary, but are bounded between constant upper and lower bounds.

Now, observe that the agents over the interpreted systems semantics perform a joint action at a given time in a global state, which means that we assume the synchronous semantics of interpreted systems. Next, in the interleaved interpreted systems only one local or shared action may be performed by agents at a given time in a global state. This means that the interleaved interpreted systems define the asynchronous semantics.

Systems can be modelled using both IIS and IS. The idea is not to convert an IS into IIS, but rather using both the representations, which are independently defined starting from a description of a system. However, for many systems an IIS model is a submodel of the corresponding IS model, (i.e., the set of states of the IIS model is a subset of the set of states of the corresponding IS model and the transition relation of an IIS model is a subset of the transition relation of the corresponding IS model), and then we can discuss the complexity of converting an IS encoding into an IIS one. In such a case, from the definitions of IS and IIS it follows that each computation of the Kripke model generated by IIS is also a valid computation of the Kripke model generated by IS. Thus, if an ELTLK formula is valid in the model generated by IIS, then this formula is also valid in the model generated by IS. However, the converse of the implication does not hold. Further, if we have a propositional formula φ that encodes the transition relation of the Kripke model generated by an IS such that the null action is enabled at each local state, then we can convert it to the formula $\varphi \wedge \varphi'$ that encodes the transition relation of the Kripke model generated by IIS and the length of φ' is $O(n \cdot log(n))$, where n is the number of the agents. The formula φ' forces the agents to work in an asynchronous way.

3 Bounded model checking

The main idea of SAT-based BMC methods consists in translating the existential model checking problem [12,48] for a modal (e.g., temporal, epistemic, deontic) logic to the propositional satisfiability problem, i.e., it consists in representing a counterexample-trace of bounded length by a propositional formula and checking the resulting propositional formula with a specialised SAT-solver. If the formula in question is satisfiable, then a satisfying assignment returned by the SAT-solver can be converted into a concrete counterexample that shows that the property is violated. Otherwise, the bound is increased and the process repeated.

Let M be a model for a system S, φ an existential formula describing a property P to be tested, and $k \in \mathbb{N}$ a bound. Moreover, let $tr_k(\varphi)$ be a propositional formula that is satisfiable if and only if the formula φ holds in the model M. Algorithm 1 shows the general SAT-based BMC approach. In Algorithm 1 we use the procedure $checkSat(\gamma)$ that for any given propositional formula γ returns one of the three possible values: SAT, UNSAT, or UNKNOWN. The meanings of the values SAT and UNSAT are self-evident. The value UNKNOWN is returned either if the procedure $checkSat$ is not able to decide the satisfiability

of its argument within some preset timeout period or has to terminate itself due to exhaustion of the memory available.

Algorithm 1 The standard SAT-based BMC algorithm

```
1: k := 0
2: loop
3:    result := checkSat(tr_k(φ))
4:    if result = SAT then
5:       return TRUE
6:    else if result = UNKNOWN then
7:       return UNKNOWN
8:    end if
      {tr_k(φ) is not satisfiable}
9:    k := k + 1
10: end loop
```

The crux of BDD-based BMC is to interleave the verification with the construction of the reachable states. Algorithm 2 illustrates a general idea of the BDD-based bounded model checking method. With $\mathcal{M}_0$ we denote the submodel that consists of the initial state of M only, and $\mathcal{M}_{\rightsquigarrow}$ denotes the model that extends the model $\mathcal{M}$ with all the immediate successors of the states of $\mathcal{M}$. At each step of the state space construction we obtain a submodel (denoted with $\mathcal{M}$) of the analysed model M, which is used to verify (line 4) the existential formula. These steps are applied repetitively until the fixed point for the state space construction is reached, i.e., $\mathcal{M} = \mathcal{M}'$, or a witness for the verified formula is found. The number of iterations needed for the algorithm to complete is counted using the variable k, which is later used in the evaluation of the approach.

Algorithm 2 General BDD-based BMC algorithm

```
1: M := M_0
2: k := 0
3: loop
4:    if verify(M, φ) = TRUE then
5:       return TRUE
6:    end if
7:    M' := M
8:    M := M_⇝
9:    if M = M' then
10:      return FALSE
11:   end if
12:   k := k + 1
13: end loop
```

3.1 BDD-based Approach

In this section we show how to perform bounded model checking for ELTLK using BDDs [12] by combining the standard approach for ELTL [11] with the method for the epistemic operators [43] similarly to the solution for CTL* of [12].

Definition 1 Let $\mathcal{PV}$ be a set of propositions. For an ELTLK formula φ 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 $\odot \in \{\neg, \mathrm{X}\}$, then $\gamma(\varphi) = \gamma(\varphi')$,
- if $\varphi = \varphi' \odot \varphi''$ and $\odot \in \{\wedge, \vee, \mathrm{U}, \mathrm{R}\}$, then $\gamma(\varphi) = \gamma(\varphi') + \gamma(\varphi'')$,
- if $\varphi = \mathrm{Y}\varphi'$ and $\mathrm{Y} \in \{\overline{\mathrm{K}}_c, \overline{\mathrm{E}}_\Gamma, \overline{\mathrm{D}}_\Gamma, \overline{\mathrm{C}}_\Gamma\}$, then $\gamma(\varphi) = \gamma(\varphi') + 1$.

Definition 2 Let $\mathrm{Y} \in \{\overline{\mathrm{K}}_c, \overline{\mathrm{E}}_\Gamma, \overline{\mathrm{D}}_\Gamma, \overline{\mathrm{C}}_\Gamma\}$. If $\varphi = \mathrm{Y}\psi$ is an ELTLK formula, by $sub(\varphi)$ we denote the immediate subformula ψ of the epistemic operator Y. Moreover, for an arbitrary ELTLK formula φ we define inductively the set $\mathcal{Y}(\varphi)$ of its subformulae in the form $\mathrm{Y}\psi$:

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

Definition 3 Let $M = (G, \iota, T, \{\sim_c\}_{c\in\mathcal{A}}, \mathcal{V})$ and $U \subseteq G$ with $\iota \in U$. The *submodel* generated by U is a tuple $M|_U = (U, \iota, T', \{\sim'_c\}_{c\in\mathcal{A}}, \mathcal{V}')$, where: $T' = T \cap U^2$, $\sim'_c = \sim_c \cap U^2$ for each $c \in \mathcal{A}$, and $\mathcal{V}' = \mathcal{V} \cap U^2$.

For ELTLKformulae φ, ψ, and ψ', by $\varphi[\psi \leftarrow \psi']$ we denote the formula φ in which every occurrence of ψ is replaced with ψ'. Let $M = (G, \iota, T, \{\sim_c\}_{c\in\mathcal{A}}, \mathcal{V})$ 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 $[[M, \varphi]] = \{g \in G_R \mid M, g \models^\exists \varphi\}$.

3.1.1 Reduction of ELTLK to ELTL

Let $M = (G, \iota, T, \{\sim_c\}_{c\in\mathcal{A}}, \mathcal{V})$ be a model, and φ an ELTLK formula. Here, we describe an algorithm for computing the set $[[M, \varphi]]$. The algorithm allows for combining any two methods for computing $[[M, \varphi]]$ for each φ being an ELTL formula, or in the form $\mathrm{Y}p$, where $p \in \mathcal{PV}$, and $\mathrm{Y} \in \{\overline{\mathrm{K}}_c, \overline{\mathrm{E}}_\Gamma, \overline{\mathrm{D}}_\Gamma, \overline{\mathrm{C}}_\Gamma\}$ (we use the algorithms from [11] and [43], respectively).

Algorithm 3 is used to compute the set $[[M, \varphi]]$. In order to obtain this set, we construct a new model M' together with an ELTL formula φ', as described in Algorithm 3, and compute the set $[[M', \varphi']]$, which is equal to $[[M, \varphi]]$. Initially φ' equals φ, which is an ELTLK formula, and we process the formula in stages to reduce it to an ELTL formula by replacing with atomic propositions all its subformulae containing epistemic operators. We begin by choosing some epistemic subformula ψ of φ', which consists of exactly one epistemic operator, and process it in two stages. First, we modify the valuation function of M' such that every state initialising some path or run along which $sub(\psi)$ holds is labelled with the new atomic proposition $p_{sub(\psi)}$, and we replace with the variable $p_{sub(\psi)}$ every occurrence of $sub(\psi)$ in ψ. In the second stage, we deal with the epistemic operators having in their scopes atomic propositions only. By modifying the valuation function of M' we label every state initialising some path or run along which the modified simple epistemic formula ψ holds with a new variable p_ψ. Similarly to the previous stage, we replace every occurrence of ψ in φ' with p_ψ. In the subsequent iterations, we process every remaining epistemic subformulae of φ' in the same way until there are no more nested epistemic operators in φ', i.e., we obtain an ELTL formula φ', and the model M' with the appropriately modified valuation function. Finally, we compute the set of all reachable states of M' that initialise at least one path or run along which φ' holds (line 13).

The correctness of the substitution used in Algorithm 3 is stated in the following lemma:

Lemma 1 *Let $M = (G, \iota, T, \{\sim_c\}_{c\in\mathcal{A}}, \mathcal{V})$ be a model over $\mathcal{PV}$, φ an ELTLK formula, and $g \in G$ some state of M. We define $M' = (G, \iota, T, \{\sim_c\}_{c\in\mathcal{A}}, \mathcal{V}')$ over $\mathcal{PV}' = \mathcal{PV} \cup \{q\}$, where q is an atomic proposition such that $q \notin \mathcal{PV}$, and $\mathcal{V}'$ is defined as follows:*

– $p \in \mathcal{V}(g')$ *iff* $p \in \mathcal{V}'(g')$ *for all* $p \in \mathcal{PV}$ *and* $g' \in G$,
– $M, g' \models^{\exists} \varphi$ *iff* $q \in \mathcal{V}'(g')$ *for all* $g' \in G$.

Then, $M', g \models^{\exists} q$ *iff* $M, g \models^{\exists} \varphi$.

Proof (Sketch) The "$\Rightarrow$" case follows directly from the definition of V'. The "$\Leftarrow$" case can be demonstrated by the induction on the length of a formula φ. The base case follows directly for the atomic propositions and their negations. In the inductive step we assume that the lemma holds for all the proper subformulae of φ, and use the definition of V', and the fact that M' contains exactly the same paths as M.

Algorithm 3 Computation of $[M, \varphi]$

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

3.1.2 BMC Algorithm

To perform bounded model checking of an ELTLK formula, we use Algorithm 4. Given a model M and an ELTLK formula φ, the algorithm checks if there exists a path or run initialised in ι on which φ holds, i.e., if $M, \iota \models^{\exists} \varphi$. For any $X \subseteq G$ by $X_{\rightsquigarrow} \stackrel{def}{=} \{g' \in G \mid (\exists g \in X)(\exists \rho \in \Pi(g))\ g' = \rho(1)\}$ we mean 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 ι. 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 ι is in the set of states from which there is a path or run on which φ 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 of reachable states, and a path or run from the initial state on which φ 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 formulated by the following theorem:

Theorem 1 *Let* $M = (G, \iota, T, \{\sim_c\}_{c \in \mathcal{A}}, \mathcal{V})$ *be a model,* Π *a set of paths and runs of* M, φ *an ELTLK formula, and* $\rho \in \Pi$ *a path or run with an evaluation position* m *such that* $m \trianglelefteq_{\rho} length(\rho)$. *Then,* $M, \rho[m] \models \varphi$ *iff there exists* $G' \subseteq G$ *such that* $\iota \in G'$, *and* $M|_{G'}, \rho[m] \models \varphi$.

Algorithm 4 Bounded model checking algorithm of φ in M

```
1: Reach := {ι}, New := {ι}
2: while New ≠ ∅ do
3:    Next := New_⇝
4:    if ι ∈ [[M|_Reach, φ]] then
5:       return TRUE
6:    end if
7:    New := Next \ Reach
8:    Reach := Reach ∪ New
9: end while
10: return FALSE
```

Proof "⇒" This way the proof is obvious as we simply take $G' = G$.
"⇐" This way the proof is more involved. It is by induction on the length of a formula φ. The base case is straightforward as the lemma follows directly for the propositional variables and their negations. Assume, the statement holds for all the proper subformulae of φ. Let $G' \subseteq G$ be a set of states such that $M|_{G'}$ contains ρ, and (*) let $m \in \mathbb{N}$ be an evaluation position such that $M|_{G'}, \rho[m] \models \varphi$.

1. Let $\varphi = \psi_1 \vee \psi_2$. By the semantics and the assumption (*), $M|_{G'}, \rho[m] \models \psi_1$ or $M|_{G'}, \rho[m] \models \psi_2$. Using the induction hypothesis and the definition of submodel (Definition 3), ρ exists also in the model M, and $M, \rho[m] \models \psi_1$ or $M, \rho[m] \models \psi_2$, thus $M, \rho[m] \models \psi_1 \vee \psi_2$.
2. Let $\varphi = \psi_1 \wedge \psi_2$. By the semantics and the assumption (*), $M|_{G'}, \rho[m] \models \psi_1$ and $M|_{G'}, \rho[m] \models \psi_2$. Using the induction hypothesis and the definition of submodel, ρ exists also in the model M. Therefore, $M, \rho[m] \models \psi_1$ and $M, \rho[m] \models \psi_2$, thus $M, \rho[m] \models \psi_1 \wedge \psi_2$.
3. Let $\varphi = \mathrm{X}\psi_1$. By the semantics and the assumption (*), $length(\rho) > m$, and $M|_{G'}, \rho[m+1] \models \psi_1$. Using the induction hypothesis and the definition of submodel, we get that ρ exists also in M, and $M, \rho[m+1] \models \psi_1$, therefore $M, \rho[m] \models \mathrm{X}\psi_1$.
4. Let $\varphi = \psi_1 \mathrm{U} \psi_2$. By the semantics and the assumption (*), there exists $k \geqslant m$, such that $M|_{G'}, \rho[k] \models \psi_2$, and $M|_{G'}, \rho[j] \models \psi_1$, for all $m \leqslant j < k$. Using the induction hypothesis and the definition of submodel, we get that ρ exists also in M. Therefore, from $M, \rho[k] \models \psi_2$, and $M, \rho[j] \models \psi_1$ for all $m \leqslant j < k$, it follows that $M, \rho[m] \models \psi_1 \mathrm{U} \psi_2$.
5. Let $\varphi = \psi_1 \mathrm{R} \psi_2$. By the semantics and the assumption (*) we have one or both of the following cases:
   (a) ρ is a path of $M|_{G'}$, and $M|_{G'}, \rho[k] \models \psi_2$ for all $k \geqslant m$, then from the definition of submodel, ρ exists also in M, and $\rho \in \Pi^{\omega}$. Using the induction hypothesis, we have that $M, \rho[k] \models \psi_2$ for all $k \geqslant m$. Therefore, it follows that $M, \rho[m] \models \psi_1 \mathrm{R} \psi_2$.
   (b) There exists $k \geqslant m$ such that $M|_{G'}, \rho[k] \models \psi_1$, and $M|_{G'}, \rho[j] \models \psi_2$ for all $m \leqslant j \leqslant k$. From the definition of submodel, ρ also exists in M, and using the induction hypothesis we get that $M, \rho[k] \models \psi_1$, and $M, \rho[j] \models \psi_2$ for all $m \leqslant j \leqslant k$. Thus, $M, \rho[m] \models \psi_1 \mathrm{R} \psi_2$.
6. Let $c \in \mathcal{A}$ and $\varphi = \overline{\mathrm{K}}_c \psi_1$. By the semantics and the assumption (*), there exists such a path or run ρ' in $M|_{G'}$ that $\rho'(k) \sim_c \rho(m)$ for some $k \geqslant 0$, and $M|_{G'}, \rho'[k] \models \psi_1$. From the definition of submodel, ρ and ρ' also exist in M. Using the induction hypothesis, we get that $M, \rho'[k] \models \psi_1$ and $\rho'(k) \sim_c \rho(m)$. Thus, $M, \rho[m] \models \overline{\mathrm{K}}_c \psi_1$.
7. Let $\Gamma \subseteq \mathcal{A}$ and $\varphi = \overline{\mathrm{Y}}_{\Gamma} \psi_1$, where $\mathrm{Y} \in \{\mathrm{D}, \mathrm{E}, \mathrm{C}\}$. By the semantics and the assumption (*), there exists such a path or run ρ' in $M|_{G'}$ that $\rho'(k) \sim_{\Gamma}^{\mathrm{Y}} \rho(m)$ for some $k \geqslant 0$,

and $M|_{G'}, \rho'[k] \models \psi_1$. From the definition of submodel, ρ and ρ' also exist in M. Using the induction hypothesis, we get that $M, \rho'[k] \models \psi_1$ and $\rho'(k) \sim^{\mathrm{Y}}_{\Gamma} \rho(i)$. Thus, $M, \rho[m] \models \overline{\mathrm{Y}}_{\Gamma}\psi_1$.

3.1.3 Model Checking ELTL

In Algorithm 3, to compute the sets of states in which ELTL formulae hold, it is possible to use any method that computes the set $[[M, \varphi]]$ for φ being an ELTL formula. The method described in [11] uses a tableau construction for which many improvements have been proposed, e.g., [15,18,19,45], but for the purpose of implementing a complete solution for the BDD-based bounded model checking of ELTLK, we use the basic symbolic model checking method of [11]. This method is based on checking the non-emptiness of Büchi automata. Given a model M and an ELTL formula φ, we begin with constructing the tableau for φ (as described in [11]), that is then combined with M to obtain their product, which contains these runs of M where φ potentially holds. Next, the product is verified in terms of the CTL model checking of EG$true$ formula under fairness constraints. Those constraints, corresponding to sets of states, allow to choose only the runs 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 guarantees that $\varphi \mathrm{U} \psi$ really holds, i.e., eliminates the runs where φ holds continuously, but ψ 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 $[[M, \varphi]]$, i.e., the reachable states where the verified formula holds. For more details, we refer the reader to [11].

The method described above has some limitations when used for bounded model checking, where it is preferable to detect counterexamples using not only the runs but also the paths of the submodel. As totality of the transition relation of the verified model is assumed, counterexamples are found only along the runs of the model. However, the method remains correct even if the final submodel only 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 our approach.

3.1.4 Model checking epistemic modalities

In the case of the formulae of the form $\mathrm{Y}p$, where $p \in \mathcal{PV}$, and $\mathrm{Y} \in \{\overline{\mathrm{K}}_c, \overline{\mathrm{E}}_{\Gamma}, \overline{\mathrm{D}}_{\Gamma}, \overline{\mathrm{C}}_{\Gamma}\}$, for the implementation purposes we use the algorithms described in [43]. The procedures simply follow from the semantics of ELTLK. The algorithm for $\overline{\mathrm{C}}_{\Gamma}$ involves a fixpoint computation, whereas for the remaining operators the algorithms are based on simple non-iterative computations.

3.2 SAT-based Approach

In this section we present two SAT-based BMC methods for ELTLK. The first one is defined for interleaved interpreted systems while the second one is defined for interpreted systems. The main difference between the two methods is in the propositional encoding of the transition relation of the model under consideration.

In SAT-based BMC we construct a propositional formula that is satisfiable if and only if there exists a finite set of paths of the underlying model that is a solution to the existential model checking problem. In order to construct the propositional formula, first we need to

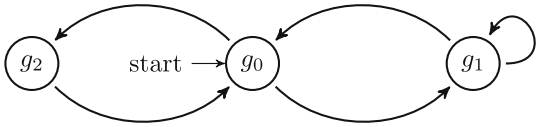


Fig. 7 A model. We assume that we have one agent that has three states: g_0, g_1 and g_2. The state g_0 is initial, and the epistemic relation is $\{(g_0 \sim g_0), (g_1 \sim g_1), (g_2 \sim g_2)\}$

define the bounded semantics for the underlying logic (i.e., in our case for ELTLK), then to encode the semantics by means of a propositional formula, and finally to represent a part of the model by a propositional formula.

The bounded semantics and the encoding for ELTLK, which is presented in this section, is based on the semantics and encoding of [55] for the temporal fragment and on the semantics and encoding of [52] for the epistemic fragment of ELTLK. This bounded semantics differs from the bounded semantics for ELTLK defined in [42] in the definition of the *k-path* that allows to replace two separate bounded semantics for k-paths that are loops and for k-paths that do not need to be loops, with one bounded semantics that is simpler, more elegant, and results in a more efficient translation of the bounded model checking problem to the SAT problem.

The propositional formula that encodes the bounded semantics for ELTLK is independent of the type of the considered model, i.e., the encoding is the same for both the interpreted systems and the interleaved interpreted systems. This encoding differs from the one defined in [42] in the definiion of the looping condition, and in using an appropriately chosen subsets of symbolic paths that are needed to encode subformulae of a formula in question.

We start with presenting the definition of the bounded semantics for ELTLK and showing that the bounded and unbounded semantics are equivalent. Then, we show a translation of the existential model checking problem for ELTLK to the propositional satisfiability problem. Finally, we prove correctness and completeness of the translation to SAT.

3.2.1 Bounded semantics for ELTLK

Let $M = (G, \iota, T, \{\sim_c\}_{c\in\mathcal{A}}, \mathcal{V})$ be a model defined for either IIS or IS, and $k \in \mathbb{N}$ a bound. A *k-path* is a pair (ρ, l), also denoted by ρ_l, where $0 \leqslant l \leqslant k$, and ρ is a finite sequence $\rho = (g_0, \ldots, g_k)$ of states such that $(g_j, g_{j+1}) \in T$ for each $0 \leqslant j < k$. A k-path ρ_l is a *loop* if $l < k$ and $\rho(k) = \rho(l)$. By $\Pi_k(g)$ we denote the set of all the k-paths ρ_l with $\rho(0) = g$. If a k-path ρ_l is a loop, then it represents the run of the form uv^ω, where $u = (\rho(0), \ldots, \rho(l))$ and $v = (\rho(l+1), \ldots, \rho(k))$. We denote this unique run by $\varrho(\rho_l)$.

To illustrate the notion of k-paths and loops, let us consider the following model shown in Fig. 7. Observe that the pairs: $\rho_0 = ((g_0, g_1, g_0, g_2, g_0), 0)$, $\rho_1 = ((g_0, g_1, g_0, g_2, g_0), 1)$, $\rho_2 = ((g_0, g_1, g_0, g_2, g_0), 2)$, $\rho_3 = ((g_0, g_1, g_0, g_2, g_0), 3)$, $\rho_4 = ((g_0, g_1, g_0, g_2, g_0), 4)$ are k-paths for $k = 4$. Moreover, only ρ_0 and ρ_2 are loops. Observe also that the k-path ρ_2 represents the following path: $(g_0, g_1)(g_0, g_2)^\omega = (g_0, g_1, g_0, g_2, g_0, g_2, g_0, g_2, \ldots)$.

As in the definition of the semantics one needs to define the satisfiability relation on suffixes of k-paths, we denote by $\rho_l[m]$ the k-path ρ_l together with the designated starting point m, where $0 \leqslant m \leqslant k$.

Definition 4 (*Bounded semantics*) Let $M = (G, \iota, T, \{\sim_c\}_{c\in\mathcal{A}}, \mathcal{V})$ be a model defined for either IIS or IS, $k \geqslant 0$ a bound, and φ an ELTLK formula. The formula φ is k—true along the k-path ρ_l (in symbols $M, \rho_l \models_k \varphi$) iff $M, \rho_l[0] \models_k \varphi$, where

$M, \rho_l[m] \models true$,
$M, \rho_l[m] \not\models false$,
$M, \rho_l[m] \models_k p$ iff $p \in \mathcal{V}(\rho(m))$,
$M, \rho_l[m] \models_k \neg p$ iff $p \notin \mathcal{V}(\rho(m))$,
$M, \rho_l[m] \models_k \varphi \vee \psi$ iff $M, \rho_l[m] \models_k \varphi$ or $M, \rho_l[m] \models_k \psi$,
$M, \rho_l[m] \models_k \varphi \wedge \psi$ iff $M, \rho_l[m] \models_k \varphi$ and $M, \rho_l[m] \models_k \psi$,
$M, \rho_l[m] \models_k \mathrm{X}\varphi$ iff $m < k$ and $M, \rho_l[m+1] \models_k \varphi$ or $m = k$ and $l < k$ and $\rho(k) = \rho(l)$ and $M, \rho_l[l+1] \models_k \varphi$,
$M, \rho_l[m] \models_k \varphi \mathrm{U} \psi$ iff $(\exists m \leqslant i \leqslant k)(M, \rho_l[i] \models_k \psi$ and $(\forall m \leqslant j < i) M, \rho_l[j] \models_k \varphi)$ or $(\rho(k) = \rho(l)$ and $l < m$ and $(\exists l < i < m)(M, \rho_l[i] \models_k \psi$ and $(\forall m \leqslant j \leqslant k) M, \rho_l[j] \models_k \varphi$ and $(\forall l \leqslant j < i) M, \rho_l[j] \models_k \varphi))$,
$M, \rho_l[m] \models_k \varphi \mathrm{R} \psi$ iff $(\forall min(l, m) \leqslant i \leqslant k)(\rho(k) = \rho(l)$ and $l < k$ and $M, \rho_l[i] \models_k \psi)$ or $(\exists m \leqslant i \leqslant k)(M, \rho_l[i] \models_k \varphi$ and $(\forall m \leqslant j \leqslant i) M, \rho_l[j] \models_k \psi)$ or $(\rho(k) = \rho(l)$ and $l < m$ and $(\exists l < i < m)(M, \rho_l[i] \models_k \varphi$ and $(\forall m \leqslant j \leqslant k) M, \rho_l[j] \models_k \psi$ and $(\forall l \leqslant j \leqslant i) M, \rho_l[j] \models_k \psi))$,
$M, \rho_l[m] \models_k \overline{\mathrm{K}}_c \varphi$ iff $(\exists \rho'_{l'} \in \Pi_k(\iota))(\exists 0 \leqslant j \leqslant k) M, \rho'_{l'}[j] \models_k \varphi)$ and $\rho(m) \sim_c \rho'(j))$,
$M, \rho_l[m] \models_k \overline{Y}_\Gamma \varphi$ iff $(\exists \rho'_{l'} \in \Pi_k(\iota))(\exists 0 \leqslant j \leqslant k)(M, \rho'_{l'}[j] \models_k \varphi)$ and $\rho(m) \sim^Y_\Gamma \rho'(j))$, where $Y \in \{\mathrm{D}, \mathrm{E}, \mathrm{C}\}$.

We use the following notation $M \models^{\exists}_k \varphi$ iff $M, \rho_l \models_k \varphi$ for some $\rho_l \in \Pi_k(\iota)$. *The SAT-based bounded model checking problem* consists in finding out whether there exists $k \in \mathbb{N}$ such that $M \models^{\exists}_k \varphi$.

Let m be a formula evaluation position, k a bound, and $p, q \in \mathcal{PV}$. An illustration of the bounded semantics is shown in Figs. 8, 9, 10, 11, 12.

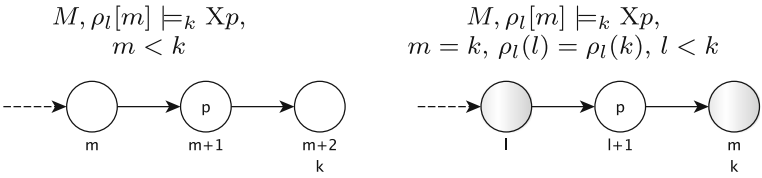


Fig. 8 Evaluation of formulae of the Next state type. The highlighted states are the same, i.e. $\rho_l(l) = \rho_l(k)$

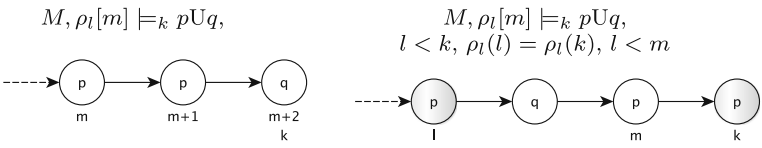


Fig. 9 Evaluation of formulae of the Until type. The highlighted states are the same, i.e. $\rho_l(l) = \rho_l(k)$

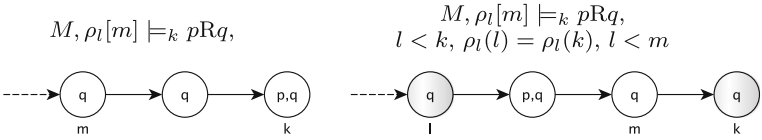


Fig. 10 Evaluation of formulae of the Release type. The highlighted states are the same, i.e. $\rho_l(l) = \rho_l(k)$

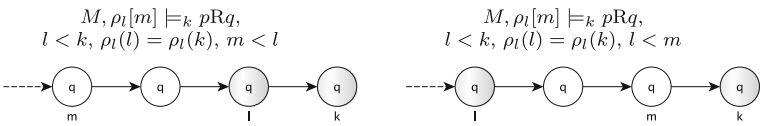


Fig. 11 Evaluation of formulae of the Release type. The highlighted states are the same, i.e. $\rho_l(l) = \rho_l(k)$

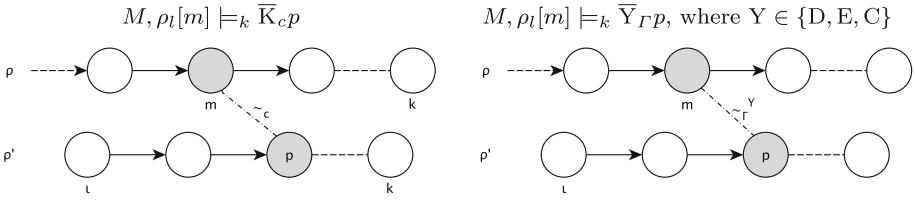


Fig. 12 Evaluation of existential epistemic formulae. The highlighted states are epistemically equivalent

3.2.2 Equivalence of the bounded and unbounded semantics

Now, we show that for some particular bound the bounded semantics is equivalent to the unbounded semantics.

Lemma 2 *Let M be a model, φ an ELTLK formula, $k > 0$ a bound, ρ_l a k-path in M, and $0 \leqslant m \leqslant k$. The following implication holds: $M, \rho_l[m] \models_k \varphi$ implies*

1. *if ρ_l is not a loop, then $M, \pi[m] \models \varphi$ for each run π in M such that $\pi[..k] = \rho$.*
2. *if ρ_l is a loop, then $M, \varrho(\rho_l)[m] \models \varphi$.*

Proof (Induction on the length of φ) The lemma follows directly for the propositional variables and their negations. Consider φ to be of the following form:

1. Let $\varphi = \psi_1 \vee \psi_2 \mid \psi_1 \wedge \psi_2 \mid \mathrm{X}\psi \mid \psi_1 \mathrm{U} \psi_2 \mid \psi_1 \mathrm{R} \psi_2$. By induction hypothesis—see Lemma 2.1. of [55].
2. $\varphi = \overline{\mathrm{K}}_c \psi$. From $M, \rho_l[m] \models_k \varphi$ it follows that $(\exists \rho'_{l'} \in \Pi_k(\iota))(\exists 0 \leqslant j \leqslant k)\ (M, \rho'_{l'}[j] \models_k \psi$ and $\rho(m) \sim_c \rho'(j))$. Assume that both ρ_l and $\rho'_{l'}$ are not loops. By inductive hypothesis, for every run π' in M such that $\pi'[..k] = \rho'$, $(\exists 0 \leqslant j \leqslant k)(M, \pi'[j] \models \psi$ and $\rho(m) \sim_c \pi'(j))$. Further, for every run π in M such that $\pi[..k] = \rho$, we have that $\pi(m) \sim_c \rho'(j)$. Thus, for every run π in M such that $\pi[..k] = \rho,\ M, \pi[m] \models \varphi$.
   Now assume that $\rho'_{l'}$ is not a loop and ρ_l is a loop. By inductive hypothesis, for every run π' in M such that $\pi'[..k] = \rho'$, $(\exists 0 \leqslant j \leqslant k)(M, \pi'[j] \models \psi$ and $\rho(m) \sim_c \pi'(j))$. Further, observe that $\varrho(\rho_l)(m) = \rho(m)$, thus $M, \varrho(\rho_l)[m] \models \varphi$.
   Now assume that both ρ_l and $\rho'_{l'}$ are loops. By inductive hypothesis, $(\exists 0 \leqslant j \leqslant k)$ $(M, \varrho(\rho'_{l'})[j] \models \psi$ and $\rho(m) \sim_c \varrho(\rho'_{l'})(j))$. Further, observe that $\varrho(\rho_l)(m) = \rho(m)$, thus $M, \varrho(\rho_l)[m] \models \varphi$.
   Now assume that $\rho'_{l'}$ is a loop, and ρ_l is not a loop. By inductive hypothesis, $(\exists 0 \leqslant j \leqslant k)(M, \varrho(\rho'_{l'})[j] \models \psi$ and $\rho(m) \sim_c \varrho(\rho'_{l'})(j))$. Further, for every run π in M such that $\pi[..k] = \rho$, we have that $\pi(m) \sim_c \varrho(\rho'_{l'})(j)$. Thus, for every run π in M such that $\pi[..k] = \rho,\ M, \pi[m] \models \varphi$.
3. Let $\varphi = \overline{Y}_\Gamma \psi$, where $Y \in \{\mathrm{D}, \mathrm{E}, \mathrm{C}\}$. These cases can be proven analogously to the case 2.

Lemma 3 (Theorem 3.1 of [5]) *Let M be a model, α an LTL formula, and ρ a run. Then, the following implication holds: $M, \rho \models \alpha$ implies that for some $k \geqslant 0$ and $0 \leqslant l \leqslant k$, $M, \pi_l \models_k \alpha$ with $\rho[..k] = \pi$.*

Lemma 4 *Let M be a model, α an LTL formula, $Y \in \{\overline{\mathrm{K}}_c, \overline{\mathrm{D}}_\Gamma, \overline{\mathrm{E}}_\Gamma, \overline{\mathrm{C}}_\Gamma\}$, and ρ a run. Then, the following implication holds: $M, \rho \models Y\alpha$ implies that for some $k \geqslant 0$ and $0 \leqslant l \leqslant k$, $M, \pi_l \models_k Y\alpha$ with $\rho[..k] = \pi$.*

Proof Let X^j denote the next-time operator applied j times, i.e., $X^j = \underbrace{X \ldots X}_{j}$.

1. Let $Y = \overline{\mathrm{K}}_c$. Then $M, \rho \models \overline{\mathrm{K}}_c\alpha$ iff $M, \rho[0] \models \overline{\mathrm{K}}_c\alpha$ iff $(\exists \rho' \in \Pi(\iota))\ (\exists j \geqslant 0)[\rho'(j) \sim_c \rho(0)$ and $M, \rho'[j] \models \alpha]$. Since $\rho'(j)$ is reachable from the initial state of M, the checking of $M, \rho'[j] \models \alpha$ is equivalent to the checking of $M, \rho'[0] \models \mathrm{X}^j\alpha$. Now since $\mathrm{X}^j\alpha$ is a pure LTL formula, by Lemma 3 we have that for some $k \geqslant 0$ and $0 \leqslant l \leqslant k$, $M, \pi'_l[0] \models_k \mathrm{X}^j\alpha$ with $\rho'[..k] = \pi'$. This implies that $M, \pi'_l[j] \models_k \alpha$ with $\rho'[..k] = \pi'$, for some $k \geqslant 0$ and $0 \leqslant l \leqslant k$. Now, since $\rho'(j) \sim_c \rho(0)$, we have $\pi'(j) \sim_c \pi(0)$. Thus, by the bounded semantics we have that for some $k \geqslant 0$ and $0 \leqslant l \leqslant k$, $M, \pi_l \models_k \overline{\mathrm{K}}_c\alpha$ with $\rho[..k] = \pi$.
2. Let $Y = \overline{\mathrm{D}}_\Gamma$. Then $M, \rho \models \overline{\mathrm{D}}_\Gamma\alpha$ iff $M, \rho[0] \models \overline{\mathrm{D}}_\Gamma\alpha$ iff $(\exists \rho' \in \Pi(\iota))(\exists j \geqslant 0)\,[\rho'(j) \sim^{\mathrm{D}}_\Gamma \rho(0)$ and $M, \rho'[j] \models \alpha]$. Since $\rho'(j)$ is reachable from the initial state of M, the checking of $M, \rho'[j] \models \alpha$ is equivalent to the checking of $M, \rho'[0] \models \mathrm{X}^j\alpha$. Now since $\mathrm{X}^j\alpha$ is a pure LTL formula, by Lemma 3 we have that for some $k \geqslant 0$ and $0 \leqslant l \leqslant k$, $M, \pi'_l[0] \models_k \mathrm{X}^j\alpha$ with $\rho'[..k] = \pi'$. This implies that $M, \pi'_l[j] \models_k \alpha$ with $\rho'[..k] = \pi'$, for some $k \geqslant 0$ and $0 \leqslant l \leqslant k$. Now, since $\rho'(j) \sim^{\mathrm{D}}_\Gamma \rho(0)$, we have $\pi'(j) \sim^{\mathrm{D}}_\Gamma \pi(0)$. Thus, by the bounded semantics we have for some $k \geqslant 0$ and $0 \leqslant l \leqslant k$, $M, \rho_l \models_k \overline{\mathrm{D}}_\Gamma\alpha$ with $\rho[..k] = \pi$.
3. Let $Y = \overline{\mathrm{E}}_\Gamma$. Since $\overline{\mathrm{E}}_\Gamma\alpha = \bigvee_{c\in\Gamma} \overline{\mathrm{K}}_c\alpha$, the lemma follows from the case 1.
4. Let $Y = \overline{\mathrm{C}}_\Gamma$. Since $\overline{\mathrm{C}}_\Gamma\alpha = \bigvee_{i=1}^{n} (\overline{\mathrm{E}}_\Gamma)^i\alpha$, where n is the size of the model M, the lemma follows from the case 3.

Lemma 5 *Let M be a model, φ an ELTLK formula, and ρ a run. Then, the following implication holds: $M, \rho \models \varphi$ implies that for some $k \geqslant 0$ and $0 \leqslant l \leqslant k$, $M, \pi_l \models_k \varphi$ with $\rho[..k] = \pi$.*

Proof (Induction on the length of φ) The lemma follows directly for the propositional variables and their negations. Assume that the hypothesis holds for all the proper subformulas of φ and consider φ to be of the following form:

1. $\varphi = \psi_1 \vee \psi_2 \mid \psi_1 \wedge \psi_2 \mid \mathrm{X}\psi \mid \psi_1 \mathrm{U} \psi_2 \mid \psi_1 \mathrm{R} \psi_2$. Straightforward by the induction hypothesis and Lemma 3.
2. Let $\varphi = Y\alpha$, and $Y, Y_1, \ldots, Y_n, Z \in \{\overline{\mathrm{K}}_c, \overline{\mathrm{D}}_\Gamma, \overline{\mathrm{E}}_\Gamma, \overline{\mathrm{C}}_\Gamma\}$. Moreover, let $Y_1\alpha_1, \ldots, Y_n\alpha_n$ be the list of all "top level" proper Y-subformulas of α (i.e., each $Y_i\alpha_i$ is a subformula of $Y\alpha$, but it is not a subformula of any subformula $Z\beta$ of $Y\alpha$, where $Z\beta$ is different from $Y\alpha$ and from $Y\alpha_i$ for $i = 1, \ldots, n$).
   If this list is empty, then α is a "pure" LTL formula with no nested epistemic modalities. Hence, by Lemma 4 we have $M, \rho \models \varphi$ implies that for some $k \geqslant 0$ and $0 \leqslant l \leqslant k$, $M, \pi_l \models_k \varphi$ with $\rho[..k] = \pi$.
   Otherwise, introduce for each $Y_i\alpha_i$ a new proposition q_i, where $i = 1, \ldots, n$. By Lemma 1, we can augment with q_i the labelling of each state s of M initialising some run along which the epistemic formula $Y_i\alpha_i$ holds, and then translate the formula α to the formula

α', which instead of each subformula $Y_i\alpha_i$ contains adequate propositions q_i. Therefore, we obtain "pure" LTL formula. Hence, by Lemma 4 we have $M, \rho \models \varphi$ implies that for some $k \geqslant 0$ and $0 \leqslant l \leqslant k$, $M, \pi_l \models_k \varphi$ with $\rho[..k] = \pi$.

The following lemma states that if we take all possible bounds into account, then the bounded and unbounded semantics are equivalent.

Lemma 6 *Let M be a model, φ an ELTLK formula. Then the following equivalence holds: $M \models^{\exists} \varphi$ iff there exists $k \geqslant 0$ such that $M \models^{\exists}_k \varphi$.*

Proof ("⇐") Follows directly from Lemma 2. ("⇒") Follows directly from Lemma 5.

3.2.3 Translation to the propositional satisfiability problem

Let $M = (G, \iota, T, \{\sim_c\}_{c\in\mathcal{A}}, \mathcal{V})$ be a model generated by IS or IIS—the encoding of global states of M is independent of the kind of considered interpreted system—and $k \in \mathbb{N}$ be a bound. Since the set of global states of M is finite, every element $g = (\ell_1, \ldots, \ell_n, \ell_e)$ of G can be encoded as a bit vector of some length r. Then, each state of M can be represented by a valuation of a vector $w = (\mathrm{w}_1, \ldots, \mathrm{w}_r)$ (called a *symbolic state*) of different propositional variables called *state variables*; further we assume that SV denotes the set of all the state variables, $SV(w)$ denotes the set of all the state variables occurring in the symbolic state w, and I_c denote the set of indexes of state variables that represent local states of agent c.

Example 1 Let $SV = \{\mathrm{w}_1, \mathrm{w}_2, \ldots\}$ be an infinite set of state variables. Consider the FTC system shown on Fig. 1 for two trains. A propositional encoding of all the local states of the two agents representing trains and an agent representing Controller is the following:

Train 1				*Train* 2			
State	Bit_2	Bit_1	*Formula*	*State*	Bit_4	Bit_3	*Formula*
$Away_1$	0	0	$\neg\mathrm{w}_1 \wedge \neg\mathrm{w}_2$	$Away_2$	0	0	$\neg\mathrm{w}_3 \wedge \neg\mathrm{w}_4$
$Wait_1$	1	0	$\neg\mathrm{w}_1 \wedge \mathrm{w}_2$	$Wait_2$	1	0	$\neg\mathrm{w}_3 \wedge \mathrm{w}_4$
$Tunnel_1$	0	1	$\mathrm{w}_1 \wedge \neg\mathrm{w}_2$	$Tunnel_2$	0	1	$\mathrm{w}_3 \wedge \neg\mathrm{w}_4$

Controller		
Location	Bit_5	*Formula*
Green	0	$\neg\mathrm{w}_5$
Red	1	w_5

Thus, given the above, it is easy to see that each state of the model of the FTC system can be represented by a valuation of a symbolic state $w = (\mathrm{w}_1, \ldots, \mathrm{w}_5)$.

Let NV denote the set of propositional variables, called the *natural variables*, such that $SV \cap NV = \emptyset$. Moreover, let $u = (\mathrm{u}_1, \ldots, \mathrm{u}_t)$ be a vector of natural variables of some length t, which we call a *symbolic number*, and $NV(u)$ denote the set of all the natural variables occurring in u. Further, let $PV = SV \cup NV$ and $V : PV \to \{0, 1\}$ be a *valuation of propositional variables* (a *valuation* for short). Each valuation induces the functions $\mathbf{S}$:

$SV^r \to \{0,1\}^r$ and $\mathbf{J} : NV^t \to \mathbb{N}$ defined in the following way:

$$\mathbf{S}((\mathrm{w}_1, \ldots, \mathrm{w}_r)) = (V(\mathrm{w}_1), \ldots, V(\mathrm{w}_r)) \tag{1}$$

$$\mathbf{J}((\mathrm{u}_1, \ldots, \mathrm{u}_t)) = \sum_{i=1}^{t} V(\mathrm{u}_i) \cdot 2^{i-1} \tag{2}$$

Now let w and w' be two symbolic states such that $SV(w) \cap SV(w') = \emptyset$, and u be a symbolic number. We recall the definitions of the following auxiliary propositional formulae:

- $I_g(w) := \bigwedge_{i=1}^{r} lit(g[i], \mathrm{w}_i)$, where $lit : \{0,1\} \times PV \to PV \cup \{\neg q \mid q \in PV\}$ is a function defined as: $lit(1,q) = q$ and $lit(0,q) = \neg q$. This formula, defined over $SV(w)$, encodes the state g of the model M.

Example 2 Consider the FTC system shown on Fig. 1 for two trains. Then, the propositional formula $I_\iota(w)$, which encodes the initial global state of the system, is defined as follows: $I_\iota(w) = \neg \mathrm{w}_1 \wedge \neg \mathrm{w}_2 \wedge \neg \mathrm{w}_3 \wedge \neg \mathrm{w}_4 \wedge \neg \mathrm{w}_5$.

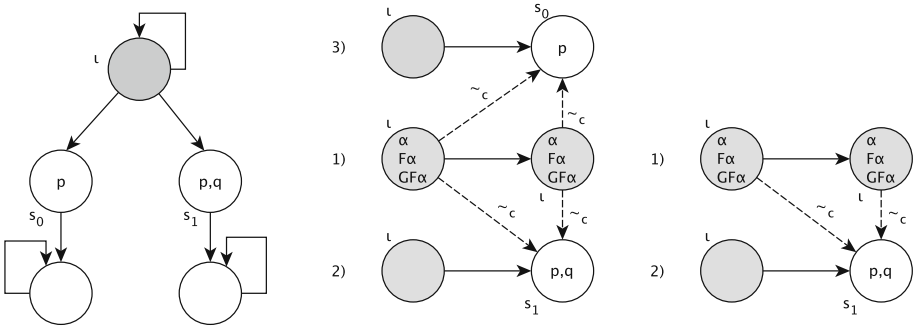
- $H(w, w') := \bigwedge_{i=1}^{r} \mathrm{w}_i \Leftrightarrow \mathrm{w}'_i$. This formula, defined over $SV(w) \cup SV(w')$, encodes equivalence between two symbolic states. It represent the fact that the symbolic states w and w' represent the same states.
- $H_c(w, w') := \bigwedge_{i \in I_c} \mathrm{w}_i \Leftrightarrow \mathrm{w}'_i$. This formula, defined over $SV(w) \cup SV(w')$, represent the fact that the local states of agent c are the same in the symbolic states w and w'.
- $p(w)$ is a formula over $SV(w)$ that is true for a valuation V iff $p \in \mathcal{V}(\mathbf{S}(w))$. This formula encodes a set of the states of M in which proposition variable $p \in \mathcal{PV}$ holds.
- $\mathcal{R}(w, w')$ is a formula over $SV(w) \cup SV(w')$ that is true for a valuation V iff $(\mathbf{S}(w), \mathbf{S}(w')) \in T$. This formula encodes the transition relation of M. The formal definition of this formula is different for M which is generated for IS and for M which is generated for IIS.
- $\mathcal{B}_j^{\sim}(u)$ is a formula over $NV(u)$ that is true for a valuation V iff $j \sim \mathbf{J}(u)$, where $\sim \in \{<, >, \leqslant, =, \geqslant\}$.

Let $M = (G, \iota, T, \{\sim_c\}_{c \in \mathcal{A}}, \mathcal{V})$ be a model, φ an ELTLK formula, and $k \geqslant 0$ a bound. We translate the problem of checking whether M is a model for φ to the problem of checking the satisfiability of the following propositional formula:

$$[M, \varphi]_k := [M^{\varphi,\iota}]_k \ \wedge \ [\varphi]_{M,k} \tag{3}$$

In order to define the formula $[M^{\varphi,\iota}]_k$ we need to specify the number of k-paths of the model M that are sufficient to validate φ. To calculate the number, we need the following auxiliary function $f_k : \mathrm{ELTLK} \to \mathbb{N}$:

- $f_k(true) = f_k(false) = f_k(p) = f_k(\neg p) = 0$, if $p \in \mathcal{PV}$,
- $f_k(\varphi \vee \psi) = max\{f_k(\varphi), f_k(\psi)\}$,
- $f_k(\varphi \wedge \psi) = f_k(\varphi) + f_k(\psi)$,
- $f_k(\mathrm{X}\varphi) = f_k(\varphi)$,
- $f_k(\varphi \mathrm{U} \psi) = k \cdot f_k(\varphi) + f_k(\psi)$,
- $f_k(\varphi \mathrm{R} \psi) = (k+1) \cdot f_k(\psi) + f_k(\varphi)$,
- $f_k(\overline{Y}\varphi) = f_k(\varphi) + 1$, for $\overline{Y} \in \{\overline{\mathrm{K}}_c, \overline{\mathrm{D}}_\Gamma, \overline{\mathrm{E}}_\Gamma\}$,
- $f_k(\overline{\mathrm{C}}_\Gamma \varphi) = f_k(\varphi) + k$.



(a) A model M **(b)** 3 different 1-paths of M. **(c)** 2 different 1-paths of M

Fig. 13 Illustration of the function f_k for $k = 1$ and the formula $\varphi = \mathrm{GF}\overline{\mathrm{K}}_c p$. In Figure **b** we assume that $\alpha = \overline{\mathrm{K}}_c p$. **a** A model M. **b** Three different 1-paths of M. **c** Two different 1-paths of M

Note that $\overline{\mathrm{C}}_\Gamma\varphi = \bigvee_{i=1}^{k}(\overline{\mathrm{E}}_\Gamma)^i\varphi$ and $f_k((\overline{\mathrm{E}}_\Gamma)^1\varphi) = f_k(\overline{\mathrm{E}}_\Gamma\varphi) = f_k(\varphi)+1$. It is easy to show, by induction on i, that $f_k((\overline{\mathrm{E}}_\Gamma)^i\varphi) = f_k(\varphi) + i$, for $i \in \{1,\ldots,k\}$. Therefore, $f_k(\overline{\mathrm{C}}_\Gamma\varphi) = f_k(\bigvee_{i=1}^{k}(\overline{\mathrm{E}}_\Gamma)^i\varphi) = max\{f_k((\overline{\mathrm{E}}_\Gamma)^1\varphi),\ldots,f_k((\overline{\mathrm{E}}_\Gamma)^k\varphi)\} = f_k((\overline{\mathrm{E}}_\Gamma)^k\varphi) = f_k(\varphi)+k$.

Now since in the BMC method we deal with the existential validity ($\models^{\exists}$), the number of k-paths sufficient to validate φ is given by the function $\widehat{f_k}$: ELTLK $\to \mathbb{N}$ that is defined as $\widehat{f_k}(\varphi) = f_k(\varphi) + 1$.

Example 3 Let $p \in \mathcal{PV}$, k be a bound. Now we calculate the number of k-paths that are sufficient to validate different ELTLK formulae.

- Let $\varphi = \mathrm{F}p$. Then, $\widehat{f_k}(\mathrm{F}p) = f_k(\mathrm{F}p) + 1 = f_k(p) + 1 = 1$; note that $\mathrm{F}\alpha = true\mathrm{U}\alpha$.
- Let $\varphi = \mathrm{GF}p$. Then, $\widehat{f_k}(\mathrm{GF}p) = f_k(\mathrm{GF}p)+1 = (k+1)\cdot f_k(\mathrm{F}p)+1 = (k+1)\cdot f_k(p)+1 = 1$; note that $\mathrm{G}\alpha = false\mathrm{R}\alpha$.
- Let $\varphi = \mathrm{GF}\overline{\mathrm{K}}_c p$. Then, $\widehat{f_k}(\mathrm{GF}\overline{\mathrm{K}}_c p) = f_k(\mathrm{GF}\overline{\mathrm{K}}_c p) + 1 = (k+1)\cdot f_k(\mathrm{F}\overline{\mathrm{K}}_c p) + 1 = (k+1)\cdot f_k(\overline{\mathrm{K}}_c p) + 1 = (k+1)\cdot(f_k(p)+1)+1 = (k+1)\cdot 1 + 1 = k+2$.
  An example of a model and a witness for the formula is shown on Fig. 13. Observe that while the value $\widehat{f_1}(\varphi)$ is 3, and the witness for φ can be of the form shown on Fig. 13b, there is a witness for φ which consists of two 1-paths only—see Fig. 13c. Thus, one can observe that the function $\widehat{f_k}$ only gives an upper bound on the number of k-paths that form a witness for an ELTLK formula.

Let $W = \{SV(w_{i,j}) \mid 0 \leqslant i \leqslant k \text{ and } 1 \leqslant j \leqslant \widehat{f_k}(\varphi)\} \cup \{NV(u_j) \mid 1 \leqslant j \leqslant \widehat{f_k}(\varphi)\}$ be a set of propositional variables. The propositional formula $[M^{\varphi,\iota}]_k$ is defined over the set W in the following way:

$$[M^{\varphi,\iota}]_k := I_\iota(w_{0,0}) \wedge \bigvee_{j=1}^{\widehat{f_k}(\varphi)} H(w_{0,0}, w_{0,j}) \wedge \bigwedge_{j=1}^{\widehat{f_k}(\varphi)} \bigwedge_{i=0}^{k-1} \mathcal{R}(w_{i,j}, w_{i+1,j}) \wedge \bigwedge_{j=1}^{\widehat{f_k}(\varphi)} \bigvee_{l=0}^{k} B_l^{=}(u_j) \tag{4}$$

where $w_{i,j}$ and u_j are, respectively, symbolic states and a symbolic number for $0 \leqslant i \leqslant k$ and $1 \leqslant j \leqslant \widehat{f_k}(\varphi)$.

Note that Formula 4 encodes $\widehat{f_k}(\varphi)$ valid k-paths of the model M that start at the initial state ι. In particular, the formula defines $\widehat{f_k}(\varphi)$ *symbolic k-paths* such that the j-th symbolic k-path $\boldsymbol{\pi}_j$ is of the form $((w_{0,j}, \ldots, w_{k,j}), u_j)$, where $w_{i,j}$ is a symbolic state for $1 \leqslant j \leqslant \widehat{f_k}(\varphi)$ and $0 \leqslant i \leqslant k$, and u_j is a symbolic number for $1 \leqslant j \leqslant \widehat{f_k}(\varphi)$.

The next step is a translation of an ELTLK formula φ to a propositional formula

$$[\varphi]_{M,k} := [\varphi]_k^{[0,1,F_k(\varphi)]} \tag{5}$$

where $F_k(\varphi) = \{j \in \mathbb{N} \mid 1 \leqslant j \leqslant \widehat{f_k}(\varphi)\}$, and $[\varphi]_k^{[m,n,A]}$ denotes the translation of φ along the n-th symbolic path $\boldsymbol{\pi}_n^m$ with the starting point m by using the set $A \subseteq F_k(\varphi)$.

For every ELTLK formula φ the function $\widehat{f_k}$ determines how many symbolic k-paths are needed for translating the formula φ. Given a formula φ and a set A of k-paths such that $|A| = \widehat{f_k}(\varphi)$, we divide the set A into subsets needed for translating the subformulae of φ. To accomplish this goal we need some auxiliary functions that were defined in [55]. We recall the definitions of these functions. First, the relation $\prec$ is defined on the power set of $\mathbb{N}$ as follows: $A \prec B$ iff for all natural numbers x and y, if $x \in A$ and $y \in B$, then $x < y$.

Now, let $A \subset \mathbb{N}$ be a finite nonempty set, and $n, d \in \mathbb{N}$, where $d \leqslant |A|$. Then,

- $g_l(A, d)$ denotes the subset B of A such that $|B| = d$ and $B \prec A \setminus B$, e.g., $g_l(\{4, 5, 6, 7, 8\}, 3) = \{4, 5, 6\}$.
- $g_r(A, d)$ denotes the subset C of A such that $|C| = d$ and $A \setminus C \prec C$, e.g., $g_r(\{4, 5, 6, 7, 8\}, 3) = \{6, 7, 8\}$.
- $g_s(A)$ denotes the set $A \setminus \{min(A)\}$, e.g., $g_s(\{4, 5, 6, 7, 8\}) = \{5, 6, 7, 8\}$.
- if n divides $|A| - d$, then $hp(A, d, n)$ denotes the sequence $(B_0, \ldots, B_n)$ of subsets of A such that $\bigcup_{j=0}^{n} B_j = A$, $|B_0| = \ldots = |B_{n-1}|$, $|B_n| = d$, and $B_i \prec B_j$ for every $0 \leqslant i < j \leqslant n$.
  Now let $h_k^{\mathrm{U}}(A, d) := hp(A, d, k)$ and $h_k^{\mathrm{R}}(A, d) := hp(A, d, k+1)$. Note that if $h_k^{\mathrm{U}}(A, d) = (B_0, \ldots, B_k)$, then $h_k^{\mathrm{U}}(A, d)(j)$ denotes the set B_j, for every $0 \leqslant j \leqslant k$. Similarly, if $h_k^{\mathrm{R}}(A, d) = (B_0, \ldots, B_{k+1})$, then $h_k^{\mathrm{R}}(A, d)(j)$ denotes the set B_j, for every $0 \leqslant j \leqslant k+1$.
  For example, if $A = \{1, 2, 3, 4, 5, 6\}$, then $h_3^{\mathrm{U}}(A, 0) = (\{1, 2\}, \{3, 4\}, \{5, 6\}, \emptyset)$, $h_3^{\mathrm{U}}(A, 3) = (\{1\}, \{2\}, \{3\}, \{4, 5, 6\})$, $h_3^{\mathrm{U}}(A, 6) = (\emptyset, \emptyset, \emptyset, \{1, 2, 3, 4, 5, 6\})$, $h_3^{\mathrm{U}}(A, d)$ is undefined for $d \in \{0, \ldots, 7\} \setminus \{0, 3, 6\}$.
  Next, $h_4^{\mathrm{R}}(A, 2) = (\{1\}, \{2\}, \{3\}, \{4\}, \{5, 6\})$, $h_4^{\mathrm{R}}(A, 6) = (\emptyset, \emptyset, \emptyset, \emptyset, \{1, 2, 3, 4, 5, 6\})$, and $h_4^{\mathrm{R}}(A, d)$ is undefined for $d \in \{0, \ldots, 7\} \setminus \{2, 6\}$.

The functions g_l and g_r are used in the translation of the formulae with the main connective being either conjunction or disjunction: for a given ELTLK formula $\varphi \wedge \psi$, if the set A is used to translate this formula, then the set $g_l(A, f_k(\varphi))$ is used to translate the subformula φ and the set $g_r(A, f_k(\psi))$ is used to translate the subformula ψ; for a given ELTLK formula $\varphi \vee \psi$, if the set A is used to translate this formula, then the set $g_l(A, f_k(\varphi))$ is used to translate the subformula φ and the set $g_l(A, f_k(\psi))$ is used to translate the subformula ψ.

The function g_s is used in the translation of the formulae with the main connective $\mathrm{Q} \in \{\overline{\mathrm{K}}_c, \overline{\mathrm{D}}_\Gamma, \overline{\mathrm{E}}_\Gamma\}$: for a given ELTLK formula $\mathrm{Q}\varphi$, if the set A is to be used to translate this formula, then the path of the number $min(A)$ is used to translate the operator Q and the set $g_s(A)$ is used to translate the subformula φ.

The function h_k^{U} is used in the translation of subformulae of the form $\varphi \mathrm{U} \psi$: if the set A is to be used to translate the subformula $\varphi \mathrm{U} \psi$ at the symbolic k-path $\boldsymbol{\pi}_n$ (with the starting point m), then for every j such that $m \leqslant j \leqslant k$, the set $h_k^{\mathrm{U}}(A, f_k(\psi))(k)$ is used to translate the

formula ψ along the symbolic path $\boldsymbol{\pi}_n$ with starting point j; moreover, for every i such that $m \leqslant i < j$, the set $h_k^{\mathrm{U}}(A, f_k(\psi))(i)$ is used to translate the formula φ along the symbolic path $\boldsymbol{\pi}_n$ with starting point i. Notice that if k does not divide $|A| - d$, then $h_k^{\mathrm{U}}(A, d)$ is undefined. However, for every set A such that $|A| = f_k(\varphi \mathrm{U} \psi)$, it is clear from the definition of f_k that k divides $|A| - f_k(\psi)$.

The function h_k^{R} is used in the translation of subformulae of the form $\varphi \mathrm{R} \psi$: if the set A is used to translate the subformula $\varphi \mathrm{R} \psi$ along a symbolic k-path $\boldsymbol{\pi}_n$ (with the starting point m), then for every j such that $m \leqslant j \leqslant k$, the set $h_k^{\mathrm{R}}(A, f_k(\varphi))(k+1)$ is used to translate the formula φ along the symbolic paths $\boldsymbol{\pi}_n$ with starting point j; moreover, for every i such that $m \leqslant i \leqslant j$, the set $h_k^{\mathrm{R}}(A, f_k(\varphi))(i)$ is used to translate the formula ψ along the symbolic path $\boldsymbol{\pi}_n$ with starting point i. Notice that if $k+1$ does not divide $|A| - 1$, then $h_k^{\mathrm{R}}(A, p)$ is undefined. However, for every set A such that $|A| = f_k(\varphi \mathrm{R} \psi)$, it is clear from the definition of f_k that $k+1$ divides $|A| - f_k(\varphi)$.

Definition 5 (*Translation of the ELTLK formulae*) Let M be a model, φ an ELTLK formula, and $k \geqslant 0$ a bound. We define inductively the translation of φ over a path number $n \in F_k(\varphi)$ starting at the symbolic state $w_{m,n}$ as shown below, where $n' = min(A)$, $h_k^{\mathrm{U}} = h_k^{\mathrm{U}}(A, f_k(\psi_2))$, and $h_k^{\mathrm{R}} = h_k^{\mathrm{R}}(A, f_k(\psi_1))$. We assume that $\mathcal{L}_k^l(\boldsymbol{\pi}_n) := \mathcal{B}_l^{=}(u_n) \wedge H(w_{k,n}, w_{l,n})$.

$$
\begin{aligned}
[true]_k^{[m,n,A]} &:= true,\\
[false]_k^{[m,n,A]} &:= false,\\
[p]_k^{[m,n,A]} &:= p(w_{m,n}),\\
[\neg p]_k^{[m,n,A]} &:= \neg p(w_{m,n}),\\
[\psi_1 \wedge \psi_2]_k^{[m,n,A]} &:= [\psi_1]_k^{[m,n,g_l(A,f_k(\psi_1))]} \wedge [\psi_2]_k^{[m,n,g_r(A,f_k(\psi_2))]},\\
[\psi_1 \vee \psi_2]_k^{[m,n,A]} &:= [\psi_1]_k^{[m,n,g_l(A,f_k(\psi_1))]} \vee [\psi_2]_k^{[m,n,g_l(A,f_k(\psi_2))]},\\
[\mathrm{X}\psi]_k^{[m,n,A]} &:= \begin{cases} [\psi]_k^{[m+1,n,A]}, & \text{if } m < k \\ \bigvee_{l=0}^{k-1}(\mathcal{L}_k^l(\boldsymbol{\pi}_n) \wedge [\psi]_k^{[l+1,n,A]}), & \text{if } m = k \end{cases}\\
[\psi_1 \mathrm{U} \psi_2]_k^{[m,n,A]} &:= \bigvee_{j=m}^{k}([\psi_2]_k^{[j,n,h_k^{\mathrm{U}}(k)]} \wedge \bigwedge_{i=m}^{j-1}[\psi_1]_k^{[i,n,h_k^{\mathrm{U}}(i)]})\\
&\quad \vee(\bigvee_{l=0}^{m-1}(\mathcal{L}_k^l(\boldsymbol{\pi}_n)) \wedge \bigvee_{j=0}^{m-1}(\mathcal{B}_j^{>}(u_n) \wedge [\psi_2]_k^{[j,n,h_k^{\mathrm{U}}(k)]}\\
&\quad \wedge \bigwedge_{i=0}^{j-1}(\mathcal{B}_i^{>}(u_n) \rightarrow [\psi_1]_k^{[i,n,h_k^{\mathrm{U}}(i)]}) \wedge \bigwedge_{i=m}^{k}[\psi_1]_k^{[i,n,h_k^{\mathrm{U}}(i)]})),\\
[\psi_1 \mathrm{R} \psi_2)]_k^{[m,n,A]} &:= \bigvee_{j=m}^{k}([\psi_1]_k^{[j,n,h_k^{\mathrm{R}}(k+1)]} \wedge \bigwedge_{i=m}^{j}[\psi_2]_k^{[i,n,h_k^{\mathrm{R}}(i)]})\\
&\quad \vee(\bigvee_{l=0}^{m-1}(\mathcal{L}_k^l(\boldsymbol{\pi}_n)) \wedge \bigvee_{j=0}^{m}(\mathcal{B}_j^{>}(u_n) \wedge [\psi_1]_k^{[j,n,h_k^{\mathrm{R}}(k+1)]}\\
&\quad \wedge \bigwedge_{i=0}^{j-1}(\mathcal{B}_i^{>}(u_n) \rightarrow [\psi_2]_k^{[i,n,h_k^{\mathrm{R}}(i)]}) \wedge \bigwedge_{i=m}^{k}[\psi_2]_k^{[i,n,h_k^{\mathrm{R}}(i)]}))\\
&\quad \vee(\bigvee_{l=0}^{k-1}(\mathcal{L}_k^l(\boldsymbol{\pi}_n)) \wedge \bigwedge_{j=0}^{m-1}(\mathcal{B}_j^{\geqslant}(u_n) \rightarrow [\psi_2]_k^{[j,n,h_k^{\mathrm{R}}(j)]})\\
&\quad \wedge \bigwedge_{j=m}^{k}[\psi_2]_k^{[j,n,h_k^{\mathrm{R}}(j)]}),\\
[\overline{\mathrm{K}}_c\psi]_k^{[m,n,A]} &:= I_\iota(w_{0,n'}) \wedge \bigvee_{j=0}^{k}([\psi]_k^{[j,n',g_s(A)]} \wedge H_c(w_{m,n}, w_{j,n'})),\\
[\overline{\mathrm{D}}_\Gamma\psi]_k^{[m,n,A]} &:= I_\iota(w_{0,n'}) \wedge \bigvee_{j=0}^{k}([\psi]_k^{[j,n',g_s(A)]} \wedge \bigwedge_{c\in\Gamma} H_c(w_{m,n}, w_{j,n'})),\\
[\overline{\mathrm{E}}_\Gamma\psi]_k^{[m,n,A]} &:= I_\iota(w_{0,n'}) \wedge \bigvee_{j=0}^{k}([\psi]_k^{[j,n',g_s(A)]} \wedge \bigvee_{c\in\Gamma} H_c(w_{m,n}, w_{j,n'})),\\
[\overline{\mathrm{C}}_\Gamma\psi]_k^{[m,n,A]} &:= [\bigvee_{j=1}^{k}(\overline{\mathrm{E}}_\Gamma)^j\psi]_k^{[m,n,A]}.
\end{aligned}
$$

For representing the propositional formula $[M, \varphi]_k$ reduced Boolean circuits (RBC) [1] are used. An RBC represents subformulae of $[M, \varphi]_k$ by fresh propositions such that

each two identical subformulae correspond to the same proposition.[1] Following van der Meyden at al. [23], instead of using RBCs, we could directly encode $[M, \varphi]_k$ in such a way that each subformula ψ of $[M, \varphi]_k$ occurring within the scope of a k-element disjunction or conjunction is replaced with a propositional variable p_ψ and the reduced formula $[M, \varphi]_k$ is conjuncted with the implication $p_\psi \Rightarrow \psi$. However, in this case our method, as the one proposed in [23], would not be complete. Nonetheless, the completeness can be achieved, by using $p_\psi \Leftrightarrow \psi$ instead of $p_\psi \Rightarrow \psi$. This however can give a formula of an exponential size during the transformation into clausal normal form. [2]

Our encoding of the ELTLK formulae is defined recursively over the structure of an ELTLK formula φ, over the current position n of the m-th symbolic k-path, and over the set A of symbolic k-paths, which is initially equal to $F_k(\varphi)$. Next, our encoding does not translate looping and non-looping witnesses separately, but it combines both of them. Further, it is parameterised by the bound k, the set of symbolic k-paths, and closely follows the bounded semantics of Def. 4. Therefore, for fixed n, m, k and A, each subformula ψ of φ requires the constraints of size $O(k \cdot f_k(\varphi))$ using the encoding of ψ at various positions. Moreover, since the encoding of a subformula ψ is only dependent on m, n, k, and A, and, multiple occurrences of the encoding of ψ over the same set of parameters can be shared, the overall size can be bounded by $O(|\varphi| \cdot k \cdot f_k(\varphi))$. Further the size of the formula $[M, \varphi]_k$ is bounded by $O(|T| \cdot k \cdot f_k(\varphi) + |\varphi| \cdot k \cdot f_k(\varphi))$.

3.2.4 Correctness and completeness of the translation

The lemmas below state the correctness and the completeness of the presented translation.

Now, let α be an ELTLK formula. For every ELTLK subformula φ of α, we denote by $[\varphi]_k^{[\alpha,m,n,A]}$ the propositional formula

$$[M]_k^{F_k(\alpha)} \wedge [\varphi]_k^{[m,n,A]} \tag{6}$$

where $[M]_k^{F_k(\alpha)} := \bigwedge_{j \in F_k(\alpha)} \bigwedge_{i=0}^{k-1} \mathcal{R}(w_{i,j}, w_{i+1,j}) \wedge \bigwedge_{j \in F_k(\alpha)} \bigvee_{l=0}^{k} B_l^{=}(u_j)$.

In the next two lemmas we use the following auxiliary notation. By $V \Vdash \xi$ we mean that the valuation V satisfies the propositional formula ξ. Moreover, we write $g_{i,j}$ instead of $\mathbf{S}(w_{i,j})$, and l_j instead of $\mathbf{J}(u_j)$.

Lemma 7 (Correctness of the translation) *Let M be a model, α an ELTLK formula, and $k \in \mathbb{N}$. For every subformula φ of the formula α, every $(m, n) \in \{0, \ldots, k\} \times F_k(\alpha)$, every $A \subseteq F_k(\alpha) \setminus \{n\}$ such that $|A| = f_k(\varphi)$, and every valuation V, the following condition holds: $V \Vdash [\varphi]_k^{[\alpha,m,n,A]}$ implies $M, ((g_{0,n}, \ldots, g_{k,n}), l_n)[m] \models_k \varphi$.*

Proof Let $n \in F_k(\alpha)$, A be a set such that $A \subseteq F_k(\alpha) \setminus \{n\}$ and $|A| = f_k(\varphi)$, m be a natural number such that $0 \leqslant m \leqslant k$, ρ_l denote the k-path $((g_{0,n}, \ldots, g_{k,n}), l_n)$, and V a valuation. Suppose that $V \Vdash [\varphi]_k^{[\alpha,m,n,A]}$ and consider the following cases:

[1] We would like to stress that we have used the RBC structure in our BMC implementations since 2003 [50], although we have not stated this explicitly in our previous works.

[2] Let α be a formula. Its clausal form is a set of clauses which is satisfiable if and only if α is satisfiable.

1. $\varphi \in \{true, false\}$. The thesis of the lemma is obvious in this case.
2. $\varphi = p$, where $p \in \mathcal{PV}$. Then, $V \Vdash [p]_k^{[\alpha,m,n,A]} \iff V \Vdash p(w_{m,n}) \iff p \in \mathcal{V}(g_{m,n}) \iff M, \rho_l[m] \models_k p$.
3. $\varphi = \neg p$, where $p \in \mathcal{PV}$. Then, $V \Vdash [\neg p]_k^{[\alpha,m,n,A]} \iff V \Vdash \neg p(w_{m,n}) \iff p \notin \mathcal{V}(g_{m,n}) \iff M, \rho_l[m] \models_k \neg p$.
4. $\varphi = \psi_1 \wedge \psi_2$. Let $B = g_l(A, f_k(\psi_1))$ and $C = g_r(A, f_k(\psi_2))$. From $V \Vdash [\psi_1 \wedge \psi_2]_k^{[\alpha,m,n,A]}$, we get $V \Vdash [\psi_1]_k^{[\alpha,m,n,B]}$ and $V \Vdash [\psi_2]_k^{[\alpha,m,n,C]}$. By inductive hypotheses, $M, \rho_l[m] \models_k \psi_1$ and $M, \rho_l[m] \models_k \psi_2$. Thus $M, \rho_l[m] \models_k \psi_1 \wedge \psi_2$.
5. $\varphi = \psi_1 \vee \psi_2$. Let $B = g_l(A, f_k(\psi_1))$ and $C = g_l(A, f_k(\psi_2))$. From $V \Vdash [\psi_1 \vee \psi_2]_k^{[\alpha,m,n,A]}$, we get $V \Vdash [\psi_1]_k^{[\alpha,m,n,B]}$ or $V \Vdash [\psi_2]_k^{[\alpha,m,n,C]}$. By inductive hypotheses, $M, \rho_l[m] \models_k \psi_1$ or $M, \rho_l[m] \models_k \psi_2$. Thus $M, \rho_l[m] \models_k \psi_1 \vee \psi_2$.
6. Let $\varphi = \mathrm{X}\psi \mid \psi_1 \mathrm{U} \psi_2 \mid \psi_1 \mathrm{R} \psi_2$ with $p \in \mathcal{PV}$. See Lemma 3.1. of [55].
7. Let $\varphi = \overline{\mathrm{K}}_c \psi$. Let $n' = \min(A)$, and $\widetilde{\rho}_{l'}$ denote the k-path $((g_{0,n'}, \ldots, g_{k,n'}), l_{n'})$. By the definition of the translation we have that $V \Vdash [\overline{\mathrm{K}}_c \psi]_k^{[\alpha,m,n,A]}$ implies $V \Vdash I_\iota(w_{0,n'}) \wedge \bigvee_{j=0}^k ([\psi]_k^{[\alpha,j,n',g_s(A)]} \wedge H_c(w_{m,n}, w_{j,n'}))$. Since $V \Vdash H_c(w_{m,n}, w_{j,n'})$ we have $g_{m,n} \sim_c g'_{j,n'}$, for some $j \in \{0, \ldots, k\}$. Therefore, by inductive hypotheses we get $(\exists 0 \leqslant j \leqslant k)$ $(M, \widetilde{\rho}_{l'}[j] \models_k \psi$ and $g_{m,n} \sim_c g'_{j,n'})$. Thus we have $M, ((g_{0,n}, \ldots, g_{k,n}), l_n)[m] \models_k \overline{\mathrm{K}}_c \psi$.
8. Let $\varphi = \overline{Y}_\Gamma \psi$, where $Y \in \{\mathrm{D}, \mathrm{E}, \mathrm{C}\}$. These cases can be proven analogously to the case 7.

Let B and C be two finite sets of indices. Then, by $Var(B)$ we denote the set of all the state variables appearing in all the symbolic states of all the symbolic k-paths whose indices are taken from the set B. Moreover, for every valuation V and every set of indices B, by $V \uparrow B$ we denote the restriction of the valuation V to the set $Var(B)$. Notice that if $B \cap C = \emptyset$, then $Var(B) \cap Var(C) = \emptyset$. This property is used in the proof of the following lemma.

Lemma 8 (Completeness of the translation) *Let M be a model, $k \in \mathbb{N}$, and α an ELTLK formula such that $f_k(\mathrm{E}\alpha) > 0$. For every subformula φ of the formula α, every $(m, n) \in \{(0,0)\} \cup \{0, \ldots, k\} \times F_k(\alpha)$, every $A \subseteq F_k(\alpha) \setminus \{n\}$ such that $|A| = f_k(\varphi)$, and every k-path ρ_l, the following condition holds: $M, \rho_l[m] \models_k \varphi$ implies that there exists a valuation V such that $\rho_l = ((g_{0,n}, \ldots, g_{k,n}), l_n)$ and $V \Vdash [\varphi]_k^{[\alpha,m,n,A]}$.*

Proof First, note that given an ELTLK formula α, and natural numbers k, m, n with $0 \leqslant m \leqslant k$ and $n \in F_k(\alpha)$, there exists a valuation V such $V \Vdash [M]_k^{F_k(\alpha)}$. This is because M has no terminal states. Now we proceed by induction on the complexity of φ.

Let $n \in F_k(\alpha)$, A be a set such that $A \subseteq F_k(\alpha) \setminus \{n\}$ and $|A| = f_k(\varphi)$, ρ_l be a k-path in M, and m be a natural number such that $0 \leqslant m \leqslant k$. Suppose that $M, \rho_l[m] \models_k \varphi$ and consider the following cases:

1. Let $\varphi = p \mid \neg p \mid \psi_1 \vee \psi_2 \mid \psi_1 \wedge \psi_2 \mid \mathrm{X}\psi \mid \psi_1 \mathrm{U} \psi_2 \mid \psi_1 \mathrm{R} \psi_2$ with $p \in \mathcal{PV}$. See the proof of Lemma 3.3. of [55].
2. Let $\varphi = \overline{\mathrm{K}}_c \psi$. Since $M, \rho_l[m] \models_k \overline{\mathrm{K}}_c \psi$, we have that $(\exists \rho'_{l'} \in \Pi_k(\iota))(\exists 0 \leqslant j \leqslant k)$ $(M, \rho'_{l'}[j] \models_k \psi)$ and $\rho(m) \sim_c \rho'(j))$. Let $n' = \min(A)$ and $B = g_s(A)$. By the inductive hypothesis and the definition of the formula H_c, there exists a valuation V' such that $V' \Vdash [M]_k^{F_k(\alpha)}$ and $V' \Vdash [\psi]_k^{[j,n',B]} \wedge H_c(w_{m,n}, w_{j,n'})$ for some $j \in \{0, \ldots, k\}$. Hence we have $V' \Vdash \bigvee_{j=0}^k ([\psi]_k^{[j,n',B]} \wedge H_c(w_{m,n}, w_{j,n'}))$. Further, since $\rho'_{l'} \in \Pi_k(\iota)$, $\rho'_{l'}(0) = \iota$. Thus, by the definition of the formula I, we get that $V' \Vdash I_\iota(w_{0,n'})$.

Therefore we have $V' \Vdash I_l(w_{0,n'}) \wedge \bigvee_{j=0}^{k}([\psi]_k^{[j,n',B]} \wedge H_c(w_{m,n}, w_{j,n'}))$, which implies that $V' \Vdash [\overline{\mathrm{K}}_c\psi]_k^{[m,n,A]}$. Since $n' \notin B$ and $n \notin A$, there exists a valuation V such that $V \uparrow B = V' \uparrow B$ and moreover $V \Vdash [M]_k^{F_k(\alpha)}$ and $V \Vdash [\overline{\mathrm{K}}_c\psi]_k^{[m,n,A]}$. Therefore we get $V \Vdash [\overline{\mathrm{K}}_c\psi]_k^{[\alpha,m,n,A]}$.

3. Let $\varphi = \overline{Y}_\Gamma\psi$, where $Y \in \{\mathrm{D}, \mathrm{E}, \mathrm{C}\}$. These cases can be proven analogously to the case 2.

The correctness of the SAT-based translation scheme for ELTLK is guaranteed by the following theorem.

Theorem 2 *Let M be a model, and φ an ELTLK formula. Then for every $k \in \mathbb{N}$, $M \models_k^{\exists} \varphi$ if, and only if, the propositional formula $[M, \varphi]_k$ is satisfiable.*

Proof ($\Longrightarrow$) Let $k \in \mathbb{N}$ and $M, \rho_l \models_k \varphi$ for some $\rho_l \in \Pi_k(\iota)$. By Lemma 8 it follows that there exists a valuation V such that $\rho_l = ((g_{0,0}, \ldots, g_{k,0}), l_0)$ with $\mathbf{S}(w_{0,0}) = g_{0,0} = \iota$ and $V \Vdash [\varphi]_k^{[\varphi,0,0,F_k(\varphi)]}$. Hence, $V \Vdash I(w_{0,0}) \wedge [M]_k^{F_k(\varphi)} \wedge [\varphi]_k^{[0,0,F_k(\varphi)]}$. Thus $V \Vdash [M^{\varphi,\iota}]_k$.
($\Longleftarrow$) Let $k \in \mathbb{N}$ and $[M^{\varphi,\iota}]_k$ be satisfiable. It means that there exists a valuation V such that $V \Vdash [M^{\varphi,\iota}]_k$. So, $V \Vdash I(w_{0,0})$ and $V \Vdash [M]_k^{F_k(\varphi)} \wedge [\varphi]_k^{[0,0,F_k(\varphi)]}$. Hence, by Lemma 7 it follows that $M, ((g_{0,0}, \ldots, g_{k,0}), l_0) \models_k \varphi$ and $\mathbf{S}(w_{0,0}) = g_{0,0} = \iota$. Thus $M \models_k^{\exists} \varphi$.

4 Experimental results

In this section we experimentally evaluate the performance of our four different BMC encodings: two SAT-based BMC (over the IIS and IS semantics) and two BDD-based BMC (over the IIS and IS semantics), all implemented as extensions of our tool VerICS [28], so the inputs to the four algorithms are the same. We compare our experimental results with these of the MCK tool (version 0.5.1),[3] the only existing tool that is suitable with respect to the input formalism (i.e., interpreted systems) and checked properties (i.e., ELTLK). We have done our best to compare our BMC approaches and the SAT-based BMC module of MCK on the same models. We would like to point out that the manual for MCK states that the tool supports SAT-based BMC for ECTL*K (i.e., ECTL* augmented to include epistemic components). Unfortunately, no theory behind this implementation has ever been published. We are aware of the paper [23], which describes SAT-based BMC for ECTLK, but it does not discuss how this approach can be extended to ECTL*K. Therefore, we are unable to compare our SAT-based BMC algorithms for ELTLK with the one for ECTL*K implemented in MCK.

We have conducted the experiments using two classical multi-agent protocols: the *(faulty) train controller system* and *the dining cryptographers protocol*, and one benchmark that is not yet so popular in the multi-agent community, i.e., the *(faulty) generic pipeline paradigm*. However, we would like to point out that (F)GPP is a very useful and scalable example, which has a potential to become a standard benchmark in this community. Further, we specify each property for the considered benchmarks in the universal form by an LTLK formula, for which we verify the corresponding counterexample formula, i.e., the negated universal formula in ELTLK which is interpreted existentially. Moreover, for every specification given, there exists

[3] http://cgi.cse.unsw.edu.au/~mck/

Table 3 The FTC system with n trains

	Verics, SAT-BMC			Verics, BDD-BMC		MCK, SAT-BMC
Formula	IS-k	IIS-k	$\widehat{f_k}$	IS-k	IIS-k	IS
φ_1	2	4	2	2	4	3
φ_2	2	4	2	2	4	3

a counterexample, i.e., the ELTLK formula specifying the counterexample holds in the model of the benchmark.

We have computed our experimental results on a computer with Intel Xeon 2 GHz processor and 4 GB of RAM, running Linux 2.6, with the default limits of 2 GB of memory and 2000 seconds. Moreover, similarly to the MCK tool, we used PicoSAT [2] to test the satisfiability of the propositional formulae generated by our SAT-based BMC encodings. Our SAT-based implementation uses PicoSAT in version 957. The implementation of the BDD-based method employs the CUDD 2.5.0 [44] library for operations on BDDs.

The first benchmark we have considered is the faulty train controller system (FTC) – see Sect. 2.4 for the description of the model. This system is scaled according to the number of trains (agents), i.e., the problem parameter n is the number of trains. The specifications (universal formulae) we consider are as follows:

- $\varphi_1 = \mathrm{G}(InTunnel_1 \rightarrow \mathrm{K}_{Train_1}(\bigwedge_{i=2}^{n} \neg InTunnel_i))$ – it expresses that whenever train one is in the tunnel, it knows that no other train is in the tunnel,
- $\varphi_2 = \mathrm{G}(\mathrm{K}_{Train_1} \bigwedge_{i=1,j=2,i<j}^{n} \neg(InTunnel_i \wedge InTunnel_j))$ – it represents that the trains are aware of the mutually exclusive access to the tunnel.

The size of the reachable state space of the FTC system is $3 \cdot (n+1) \cdot 2^{n-2}$, for $n \geqslant 2$. The sizes of the counterexamples for the above formulae, and for all our BMC methods, as well as for MCK are shown in Table 3.

We would like to point out that in the case of the SAT-based BMC by size we mean the length of the k-path in the counterexample (i.e., the value k) multiplied by the number of k-paths (i.e., the value of the function $\widehat{f_k}$). In the case of the BDD-based BMC by size we mean the number of full iterations needed to find the counterexample. In Tables 3, 4, 5 we denote by IS-k and IIS-k, respectively, the minimal value of the bound in BMC that yields a counterexample for the IS and IIS semantics.

The second benchmark we have considered is the faulty generic pipeline paradigm (FGPP)—see Sect. 2.4 for the description of the model. This system is scaled according to the number of its Nodes (agents), i.e., the problem parameter n is the number of Nodes. The specifications (universal formulae) we consider are as follows:

- $\varphi_1 = \mathrm{G}(ProdSend \rightarrow \mathrm{K}_C\mathrm{K}_P ConsReady)$—it states that if Producer produces a commodity, then Consumer knows that Producer knows that Consumer has not received the commodity.
- $\varphi_2 = \mathrm{G}(Problem_n \rightarrow (\mathrm{F}Repair_n \vee \mathrm{G}Alarm_n Send))$—it expresses that each time a problem occurs at node n, then either it is repaired, or the alarm of node n is enabled.
- $\varphi_3 = \bigwedge_{i=1}^{n} \mathrm{G}(Problem_i \rightarrow (\mathrm{F}Repair_i \vee \mathrm{G}Alarm_i Send))$—it expresses that each time a problem occurs at a node, then either it is repaired or the alarm is on.
- $\varphi_4 = \bigwedge_{i=1}^{n} \mathrm{GK}_P(Problem_i \rightarrow (\mathrm{F}Repair_i \vee \mathrm{G}Alarm_i Send))$—it expresses that Producer knows that each time a problem occurs at a node, then either it is repaired or the alarm is on.

Table 4 The FGPP system with n nodes

	Verics, SAT-BMC			Verics, BDD-BMC		MCK, SAT-BMC
Formula	IS-k	IIS-k	$\widehat{f_k}$	IS-k	IIS-k	IS
φ_1	$2n+2$	$2n+2$	3	$2n+2$	$2n+2$	$6n+4$
φ_2	$2n+2$	$2n+4$	1	$2n+1$	$2n+3$	$6n-1$
φ_3	4	6	1	3	5	8
φ_4	4	6	2	3	5	5

Table 5 The DC system with n cryptographers

	Verics, SAT-BMC			Verics, BDD-BMC		MCK, SAT-BMC
Formula	IS-k	IIS-k	$\widehat{f_k}$	IS-k	IIS-k	IS
φ_1	$n+4$	$4n+1$	n	$n+4$	$4n+1$	7
φ_2	0	0	2	2	$4n+1$	1
φ_3	$n+4$	$4n+1$	$n+1$	$n+4$	$4n+1$	7

The size of the reachable state space of the FGPP system is $4 \cdot 3^{2n}$, for $n \geqslant 1$. The sizes of the counterexamples for the above formulae, and for all our BMC methods, as well as for MCK are shown in Table 4.

The third benchmark we have considered is the dining cryptographers protocol (DC)—see Sect. 2.4 for the description of the model. This system is scaled according to the number of cryptographers, i.e., the problem parameter n is the number of cryptographers (together with the coins and the oracles). The specifications (universal formulae) we consider are as follows:

- $\varphi_1 = \mathrm{G}(odd \wedge \neg paid_1 \rightarrow \bigvee_{i=2}^{n} \mathrm{K}_1(paid_i))$—it expresses that always when the number of uttered differences is odd, and the first cryptographer has not paid for dinner, then he knows which cryptographer has.
- $\varphi_2 = \mathrm{G}(\neg paid_1 \rightarrow \mathrm{K}_1(\bigvee_{i=2}^{n} paid_i))$—it states that it is always true that if the first cryptographer has not paid for dinner, then he knows that some other cryptographer has.
- $\varphi_3 = \mathrm{G}(odd \rightarrow \mathrm{C}_{\{1,\ldots,n\}} \neg(\bigvee_{i=1}^{n} paid_i))$—it 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 size of the reachable state space of the system is $3^n + (n+1) \cdot 2^n \cdot (n+1+\sum_{k=1}^{n} 2 \cdot 3^{n-k} \cdot k)$ for $n \geqslant 3$. The sizes of the counterexamples for the above formulae, and for all our BMC methods, as well as for MCK are shown in Table 5.

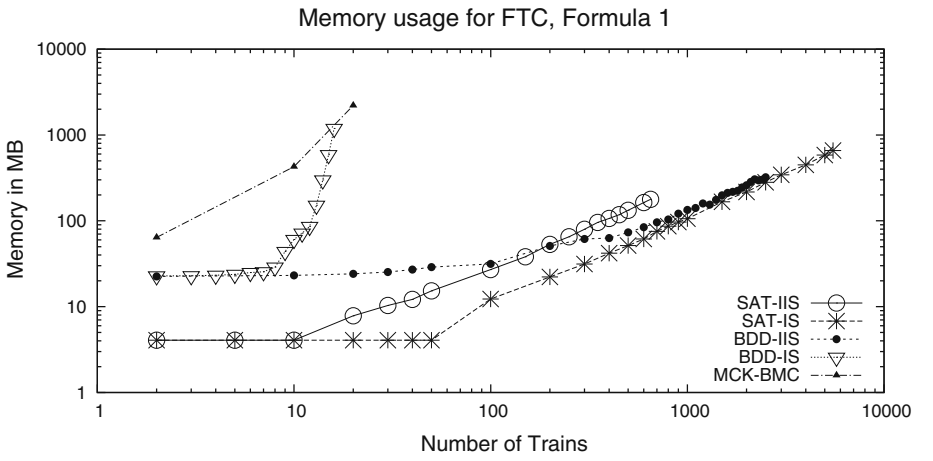
4.1 Performance evaluation

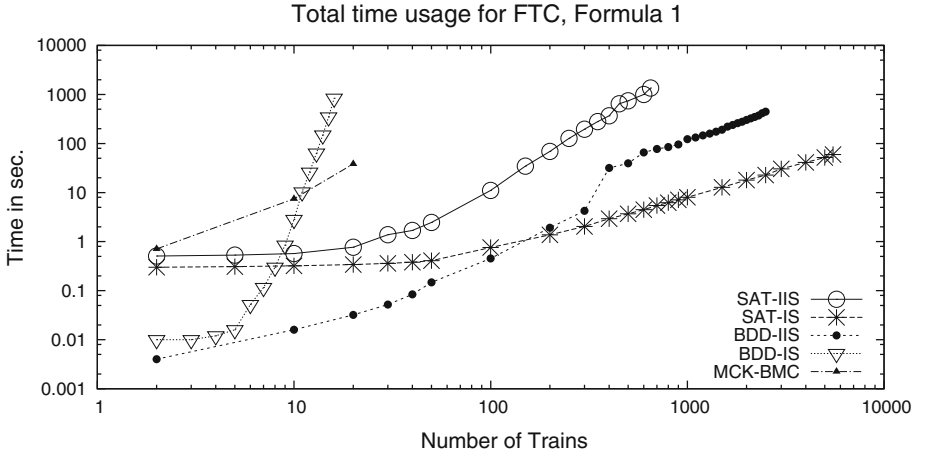
The experimental results show that the SAT-based BMC with the IS semantics outperforms the SAT-based BMC with the IIS semantics in both the memory consumption and the execution

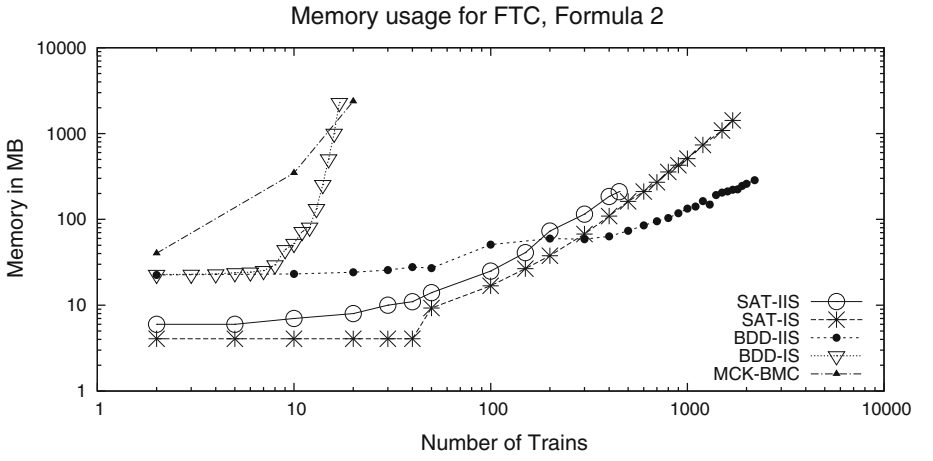
time (as shown below in the line charts), but for the BDD-based BMC this is the other way around. The reason for this is that the SAT-based BMC with the IS semantics produces a significantly smaller set of clauses (as shown in Table 6), and the SAT solver is given this smaller set. Moreover, the produced set of clauses by the SAT-based BMC with the IS semantics is not only smaller, but also 'easier' for the SAT solver, which further boosts the performance of the SAT-based BMC method with the IS semantics. The reason for the inferiority of the BDD-based BMC with the IS semantics in all of our results most likely follows from the fact that in the IS semantics, the BDD-based approach is faced with larger sets of successors in each iteration, compared to the IIS case.

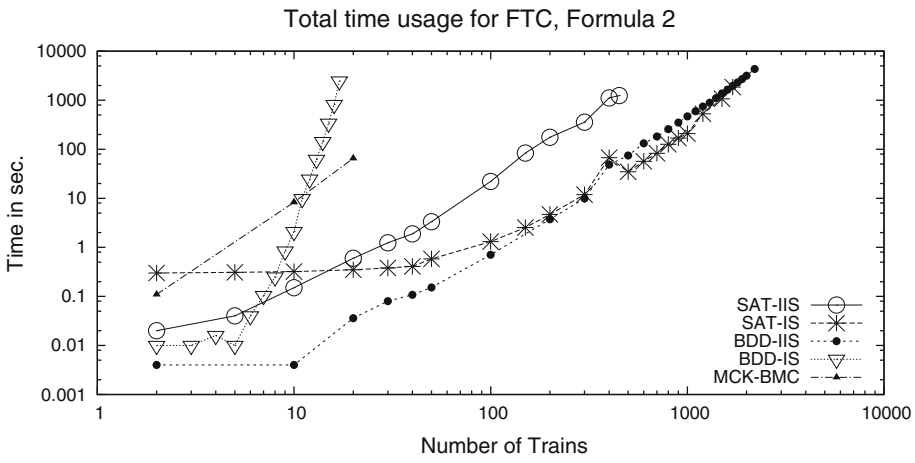
Table 6 Results for selected witnesses generated by the SAT-based BMC translations

Formula	Semantics	(Max$^{\nabla}$) nr of components	Length of the witness	Nr of paths	Nr of variables	Nr of clauses
Faulty train controller						
φ_1	IIS	650^{∇}	4	2	619982	1677373
φ_1	IS	650	2	2	250690	618440
φ_1	IS	5500^{∇}	2	2	2564618	6262036
φ_2	IIS	450^{∇}	4	2	937878	2687061
φ_2	IS	450	2	2	473350	1331220
φ_2	IS	1800^{∇}	2	2	5623947	16452621
Faulty generic pipeline paradigm						
φ_1	IIS	30^{∇}	62	3	1024009	2869312
φ_1	IS	30	62	3	844630	2257822
φ_1	IS	40^{∇}	82	3	1476472	3919425
φ_2	IIS	35^{∇}	74	1	517280	1449202
φ_2	IS	35	72	1	390327	1044692
φ_2	IS	55^{∇}	112	1	979275	2608936
φ_3	IIS	1200^{∇}	6	1	1647007	4261015
φ_3	IS	1200	6	1	2100292	5772169
φ_3	IS	1300^{∇}	6	1	1838281	4771037
φ_4	IIS	1100^{∇}	6	2	3886556	10690351
φ_4	IS	1100	6	2	3033586	7868443
φ_4	IS	1200^{∇}	6	2	3253362	8400171
Dining cryptographers						
φ_1	IIS	6^{∇}	25	6	551041	1639542
φ_1	IS	6	9	6	122437	348178
φ_1	IS	16^{∇}	19	16	2473680	7083283
φ_2	IIS	2300^{∇}	0	2	508521	793923
φ_2	IS	2300	0	2	80601	131343
φ_2	IS	2350^{∇}	0	2	82351	134193
φ_3	IIS	5^{∇}	21	22	2014710	6344695
φ_3	IS	5	9	10	267628	805315
φ_3	IS	11^{∇}	15	16	2167850	6635325


Memory usage for FTC, Formula 1
Memory in MB
Number of Trains
SAT-IIS
SAT-IS
BDD-IIS
BDD-IS
MCK-BMC


Total time usage for FTC, Formula 1
Time in sec.
Number of Trains
SAT-IIS
SAT-IS
BDD-IIS
BDD-IS
MCK-BMC


Memory usage for FTC, Formula 2
Memory in MB
Number of Trains
SAT-IIS
SAT-IS
BDD-IIS
BDD-IS
MCK-BMC

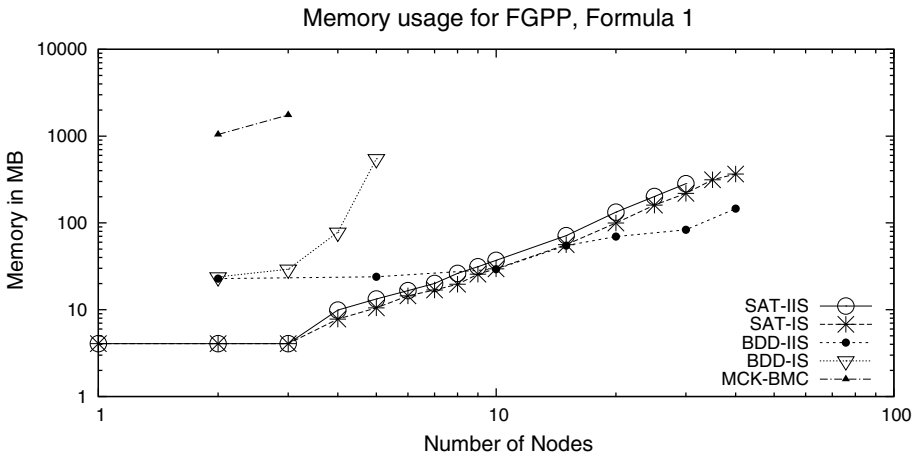


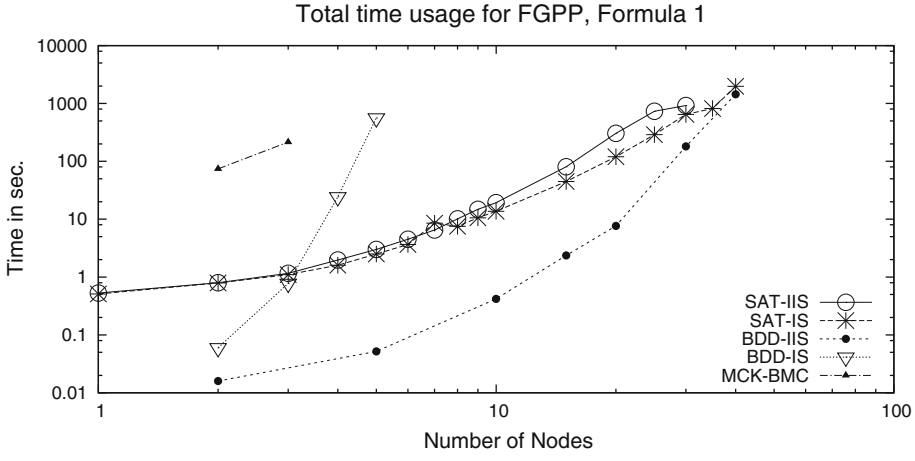
As one can see from the line charts for the FTC system, in the case of this benchmark over the IIS semantics, the BDD-based BMC performs much better in terms of the total time and the memory consumption for the formula φ_1. More precisely, in the time limit set for the benchmarks, the BDD-based BMC is able to verify the formula φ_1 for 2,500 trains, while the SAT-based BMC can handle 650 trains only. For φ_2 the BDD-based BMC is still more efficient—it is able to verify 1,700 trains, whereas the SAT-based BMC verifies only 450 trains. However, in the case of the IS semantics the SAT-based BMC is superior to the BDD-based BMC for all the tested formulae. Namely, in the set time limit, the SAT-based BMC is able to verify the formula φ_1 for 5,500 trains, while BDD-based BMC can handle 16 trains only.

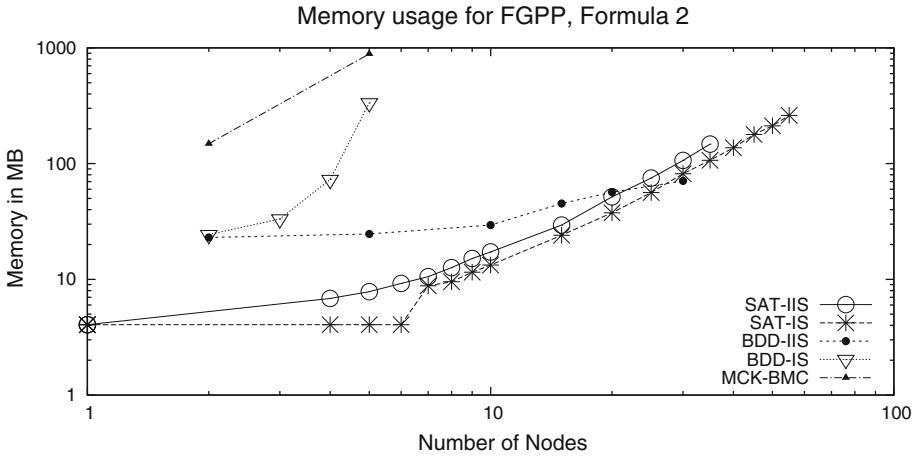
Similarly, in the case of the formula φ_2 the SAT-based BMC is able to verify 1,800 trains, while BDD-based BMC computes the results for 16 trains only.

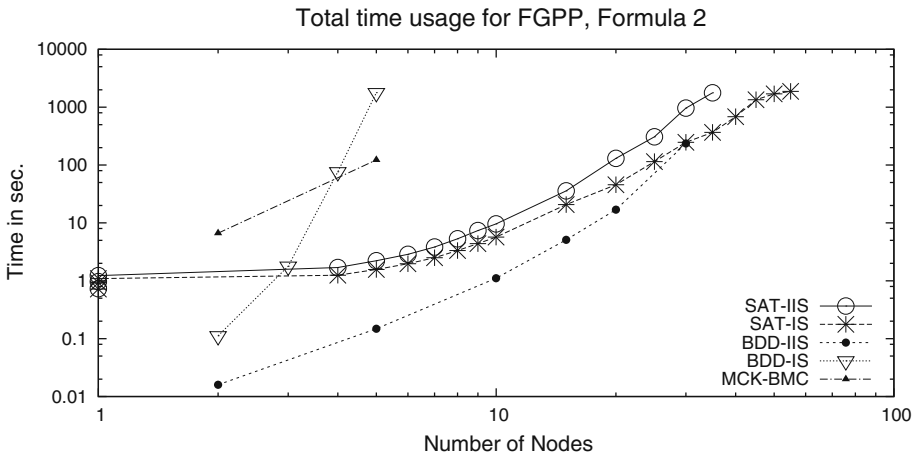
As one can see from the line charts for the FGPP system, in the case of this benchmark over the IIS semantics the SAT-based BMC performs much better in terms of the total time and the memory consumption for the formulae φ_2, φ_3, and φ_4, but it is worse for the formula φ_1. More precisely, in the set time limit, the SAT-based BMC is able to verify the formulae φ_2, φ_3 and φ_4, respectively, for 35, 1200, and 1100 nodes, while the BDD-based BMC has computes the results, respectively, for 30, 10, and 600 nodes only. In the case of the formula φ_1 the BDD-based BMC is able to verify the formula for 40 nodes, whereas the SAT-based BMC can verify this formula for 30 nodes only. Here, the reason for a higher efficiency of the BDD-based BMC is the presence of the knowledge operator that causes the partitioning of the problem to several smaller ELTL verification problems, which are handled much better by the operations on BDDs. The reason for a higher efficiency of the SAT-based BMC for the formulae φ_2, and φ_3 is the translation which uses only one symbolic k-path, whereas a higher efficiency for the formula φ_4 results from the constant length of the counterexample.

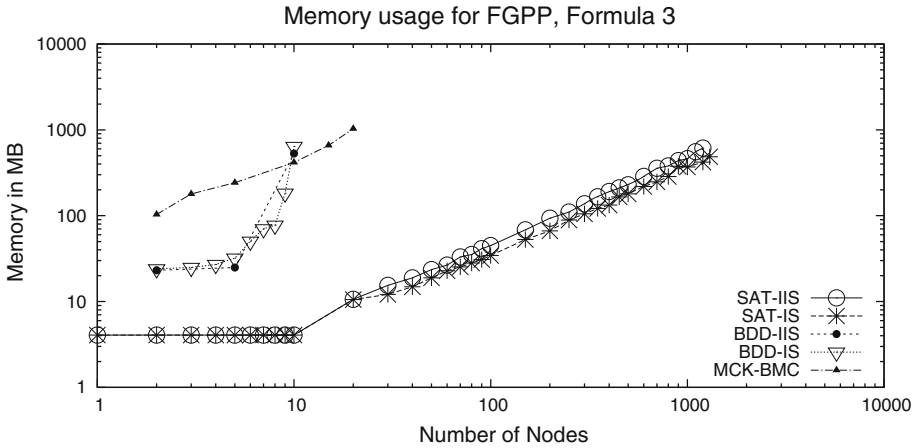
As far as the FGPP system under the IS semantics is considered, the SAT-based BMC is superior to BDD-based BMC for all the tested formulae. Namely, in the set time limit, the SAT-based BMC is able to verify the formulae φ_1, φ_2, φ_3 and φ_4, respectively, for 40, 55, 1300 and 1200 nodes, while BDD-based BMC computes the results, respectively, for 6, 5, 9 and 13 nodes only.

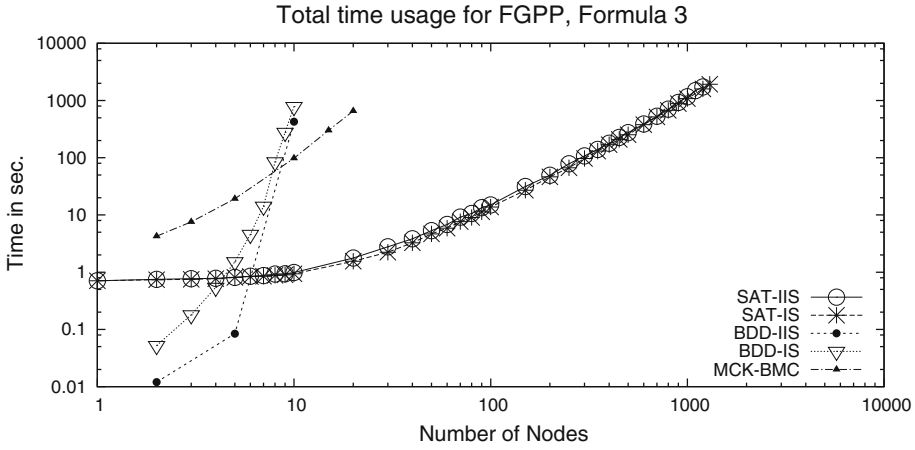

Memory usage for FGPP, Formula 1
Memory in MB
Number of Nodes
10000
1000
100
10
1
1
10
100
SAT-IIS
SAT-IS
BDD-IIS
BDD-IS
MCK-BMC

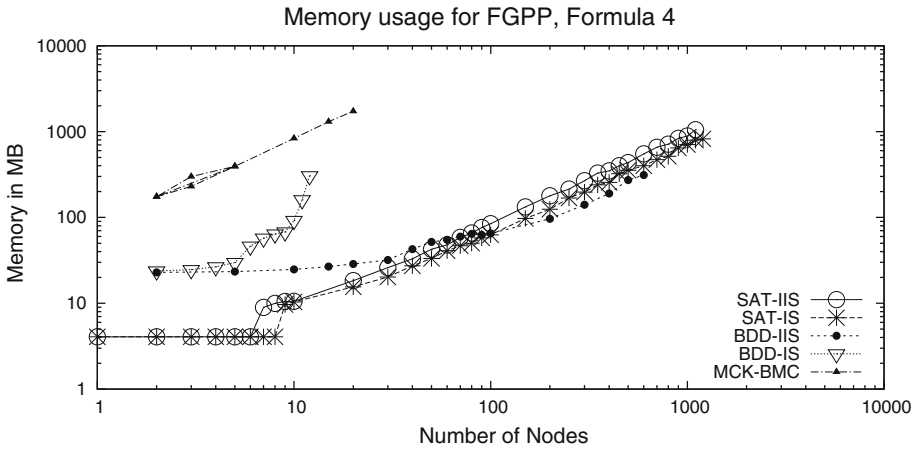

Total time usage for FGPP, Formula 1
Time in sec.
Number of Nodes
10000
1000
100
10
1
0.1
0.01
1
10
100
SAT-IIS
SAT-IS
BDD-IIS
BDD-IS
MCK-BMC

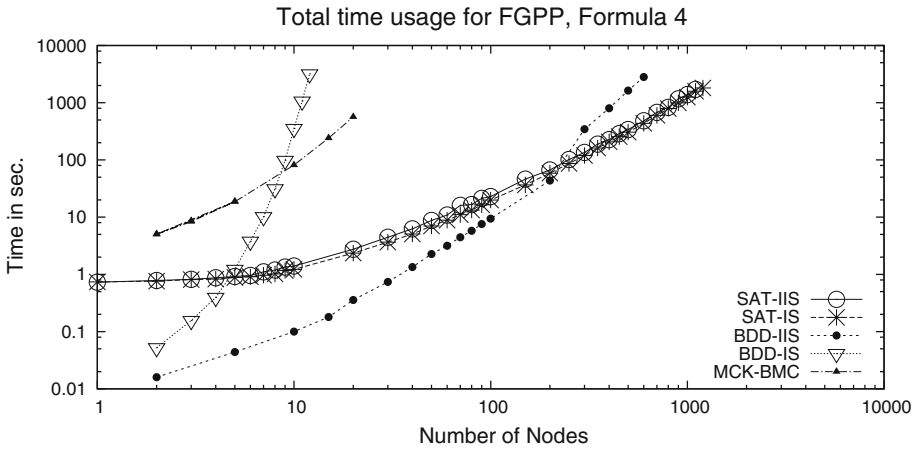

Memory usage for FGPP, Formula 2
Memory in MB
Number of Nodes
1000
100
10
1
1
10
100
SAT-IIS
SAT-IS
BDD-IIS
BDD-IS
MCK-BMC


Total time usage for FGPP, Formula 2
Time in sec.
Number of Nodes
SAT-IIS
SAT-IS
BDD-IIS
BDD-IS
MCK-BMC


Memory usage for FGPP, Formula 3
Memory in MB
Number of Nodes
SAT-IIS
SAT-IS
BDD-IIS
BDD-IS
MCK-BMC


Total time usage for FGPP, Formula 3
Time in sec.
Number of Nodes
SAT-IIS
SAT-IS
BDD-IIS
BDD-IS
MCK-BMC



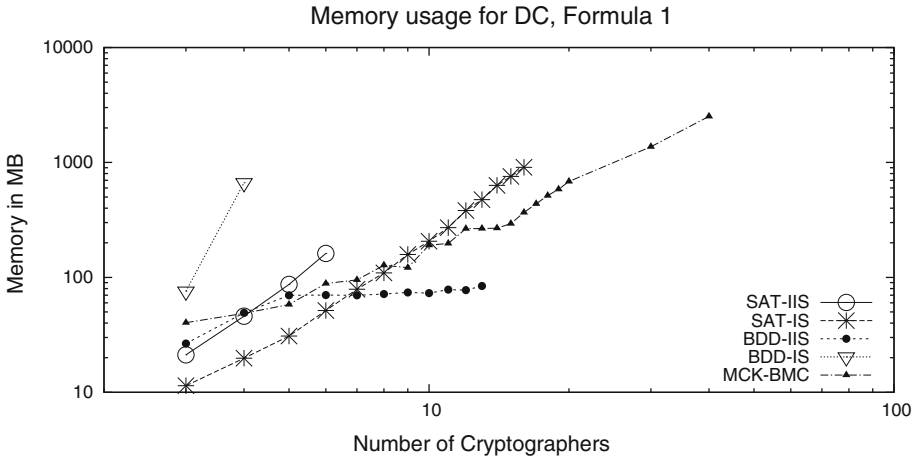


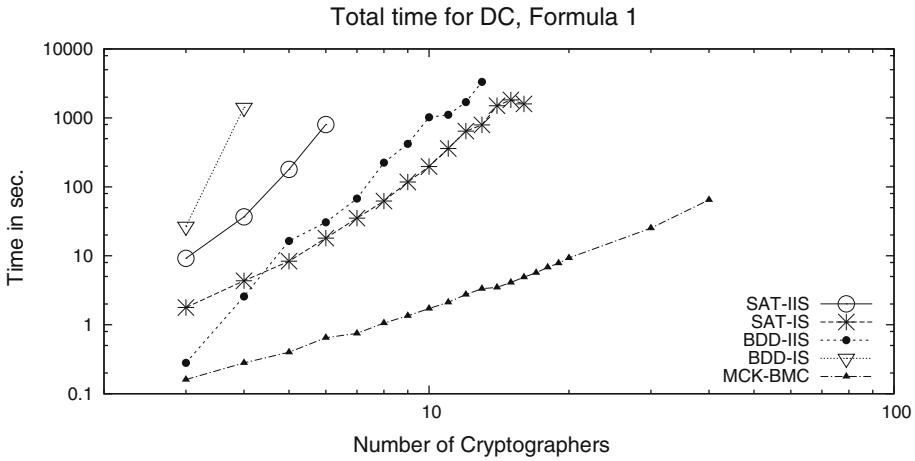
As one can see from the line charts for the DC system, in the case of this benchmark over the IIS semantics the BDD-based approach significantly outperforms the SAT-based BMC for the formulae φ_1 and φ_3, but for the formula φ_2 this is the other way around. Namely, in the set time limit, the BDD-based BMC is able to verify the formulae φ_1 and φ_3 for 12 cryptographers, while SAT-based BMC computes the results, respectively, for 6 and 5 cryptographers only. In the case of formula φ_2 SAT-based BMC computes the results for 2,300 cryptographers, whereas BDD-based BMC for 15 only.

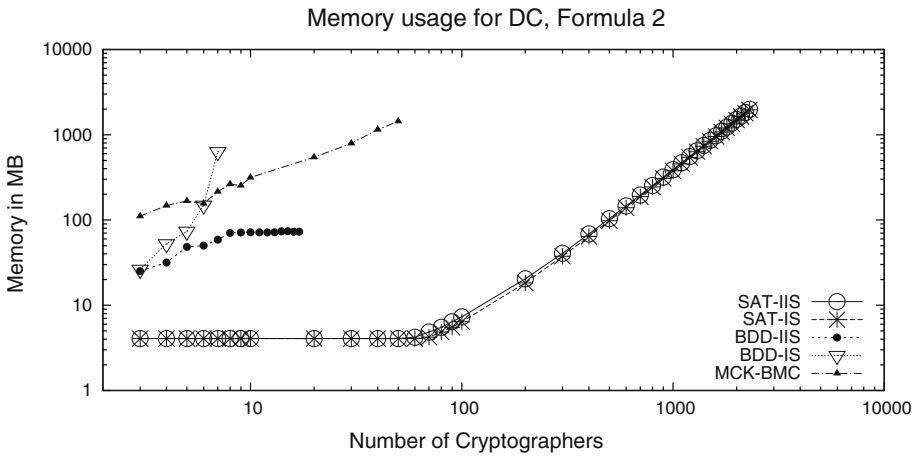
For the formulae φ_1 and φ_3 the reason of a higher efficiency of the BDD-based BMC is that the SAT-based BMC deals with a huge number of symbolic k-paths. In the case of φ_1 this number results from the fact that φ_1 contains the disjunction of the knowledge operators, whereas in the case of φ_3 the huge number of symbolic k-paths follows from the fact that φ_3 contains the common knowledge operator. A noticeable superiority of the SAT-based BMC for φ_2 follows from the following two facts: (1) the length of the SAT counterexample is constant and very small, and (2) a small number of symbolic paths in the SAT counterexample (only 2 symbolic k-paths).

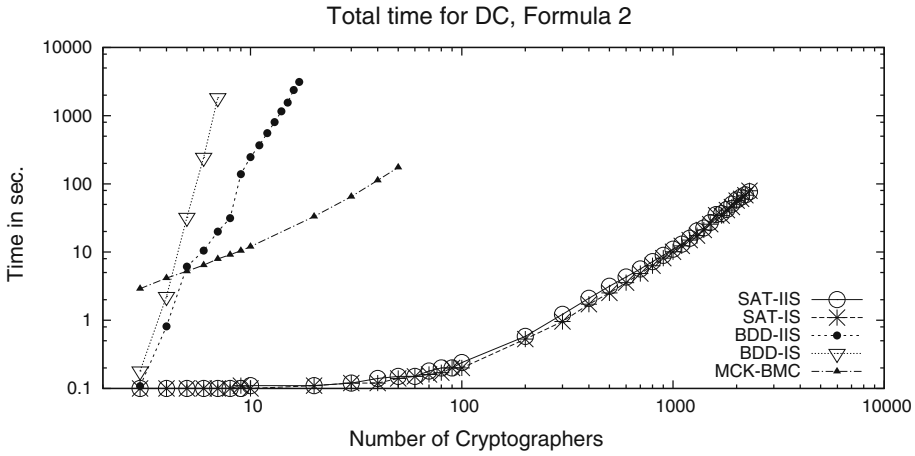
As fas as the DC system under the IS semantics is considered, the SAT-based BMC is superior to BDD-based BMC for all the tested formulae. Namely, in the set time limit, the SAT-based BMC is able to verify the formulae φ_1, φ_2, and φ_3, respectively, for 16, 2,350

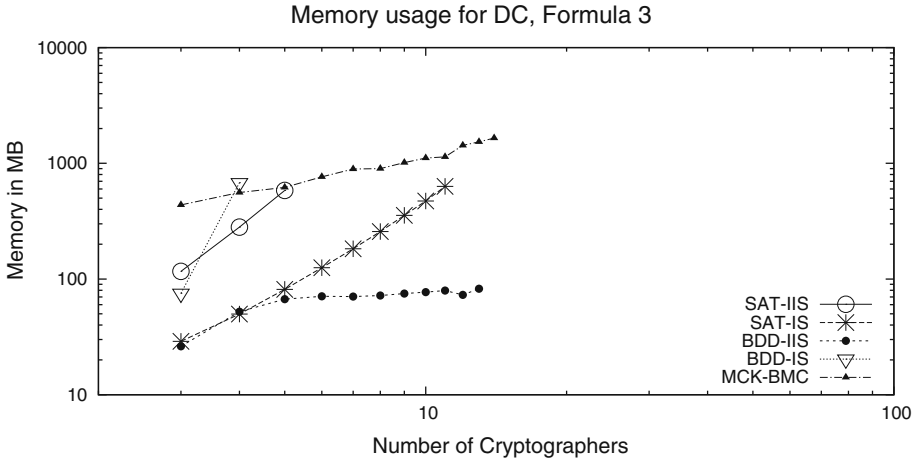
and 11 cryptographers, while BDD-based BMC computes the results, respectively, for 4, 7 and 4 cryptographers only.

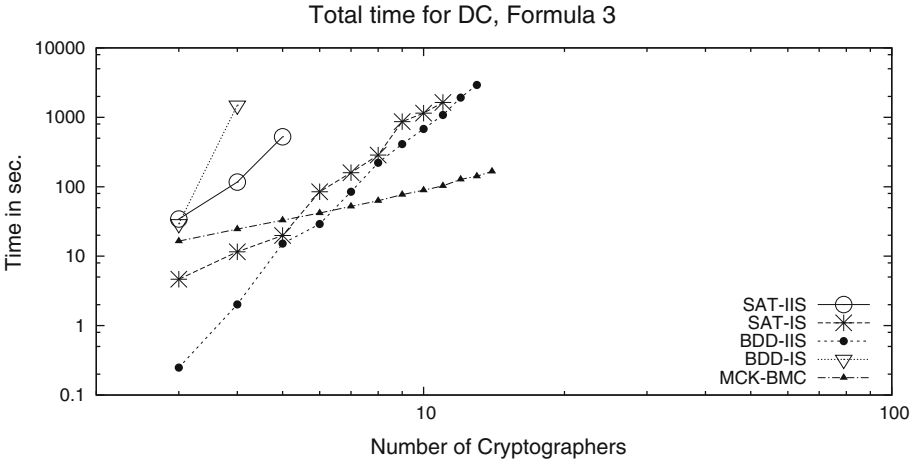







Total time for DC, Formula 2
Time in sec.
Number of Cryptographers
10000
1000
100
10
1
0.1
10
100
1000
10000
SAT-IIS
SAT-IS
BDD-IIS
BDD-IS
MCK-BMC


Memory usage for DC, Formula 3
Memory in MB
Number of Cryptographers
10000
1000
100
10
10
100
SAT-IIS
SAT-IS
BDD-IIS
BDD-IS
MCK-BMC


Total time for DC, Formula 3
Time in sec.
Number of Cryptographers
10000
1000
100
10
1
0.1
10
100
SAT-IIS
SAT-IS
BDD-IIS
BDD-IS
MCK-BMC

For the IIS semantics, the reordering of the BDD variables does not cause any improvement of the performance in the case of the benchmarks FTC and FGPP, but for the benchmark DC it reduces the memory consumption. This means that 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 the memory consumption, is also not significant and is often worth the tradeoff. Therefore, in the results for IIS we include only the BDD-based BMC variant using automatic reordering of the variables. In the case of the IS semantics the fixed interleaving order appears to be more efficient than the used reordering method. For this reason, we include only the results for the fixed interleaving order.

From our analyses we can conclude that the BDD-based BMC method is more efficient when verifying systems with the IIS semantics, whereas the SAT-based BMC method is superior when used with systems with the IS semantics. Moreover, in most cases, the BDD-based BMC spends a considerable amount of time on encoding the system, whereas the SAT-based BMC on verifying the formula. Therefore, the BDD-based BMC may provide additional time gains when verifying multiple specifications of the same system.

4.1.1 Comparison with MCK

While MCK enables verification of LTLK properties and implements the semantics of IS, it differs from our approaches in the way in which the systems are specified. We carefully inspected how the systems are represented in MCK and what a state is composed of, using the feature of printing out the state space for explicit-state reachability analysis, and noticed that the differences with our modelling are not merely syntactic. The state space is constructed by MCK in a significantly different way, for example a program counter is added for each agent, and channels are the standard way of inter-process communication.

Taking the above facts into account, we have found it not to be justified to get the numbers of states exactly equal to the ones reported by our tools. Reaching this aim could be not possible at all or would require to specify examples for MCK in an unnatural way, possibly penalising the performance. Instead, we have done our best to model the benchmarks in MCK in a way as close as possible to our approach, but modelling similarly to the ones distributed with MCK and available at the MCK web page. To this aim we have used the observable semantics while dealing with the knowledge of agents as opposed to the perfect recall semantics, which is also available in MCK.

Next, we have modelled concurrent executions in the analysed systems by means of the message-passing communication instead of the hand-shake communication. The reason is that in the message-passing communication model the protocol specification for an agent allows to have a communication channel as an argument, which enables establishing a two-point communication. Based on the knowledge available to the user, a corresponding construction for the hand-shaking approach is unsupported by MCK as an agent identifier cannot be used as an argument in the protocol definition. The hand-shaking communication is used in MCK example benchmarks and in the documentation for unscalable systems only. In the Dining Cryptographers code available at the MCK web page, the message-passing communication approach is used.

Therefore, forcing the hand-shaking communication model in MCK for our benchmarks would be very unnatural and clearly cause a performance penalty. Further, we have 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. Of course, we restrict our comparisons to the IS case. While we possibly could force the IIS semantics in the IS systems, this would be inefficient.

In the comparison of MCK with our methods, the lengths of counterexamples behave similarly, i.e. either unfold to the depth proportional to the benchmark parameter or have a fixed number of steps (with the exception of the DC model, what is described below), thus minimising the factor played by different communication schemes. These lengths are in general not equal, and do not scale in the exactly the same way, what can be seen especially for formulae φ_1 and φ_2 for FGPP. This may have two reasons: the way in which the model description is translated into the model itself, and the encoding for checking the requested properties. We can say little about the latter as no detailed counterexamples are produced by the tool. Concerning the former, we figured out by looking into the structure of the model reported for simple reachability properties that the bigger lengths are caused by a different approach to specifying systems. For example, a synchronous change of state for several components is performed in one step in our approaches, as variable values are represented by interpreted system states. On the contrary, in MCK communications via channels as well as testing and assigning of variables result in more steps. Additionally, sending and receiving messages combined with reading and assigning variables can possibly result in several values of a program counter. The comparison shows that for FGPP and FTC our BDD-BMC and SAT-BMC are superior to MCK for all the tested formulae (sometimes by several orders of magnitude). MCK consumes all the available memory even when the formulae are surprisingly small (approx. 10^6 clauses and 10^5 variables) compared to those successfully tested in our SAT-based BMC experiments (more than 10^8 clauses and variables in some cases).

An additional comment is required for the DC benchmark, where for the formulae φ_1 and φ_3, there are differences in the length of counterexamples: constant for MCK and linear for our methods. This can be traced back to the presence of the counter. In our modelling, the counter works sequentially. It introduces some limited concurrency as its actions can interleave with the preceding actions of cryptographers (to the limited degree, because the order of counting cryptographers is fixed). In MCK, there is an XOR operation available, computed in a single step. We have decided not to add a sequential counter in this case, finding it unnatural. However, it should be noted that the models are not the same for MCK and our tools for the DC benchmark, what influences the efficiency when they are explored to the full length (the diamater of the model).

The general conclusion is that while our methods can be found to be much more efficient, MCK offers a much richer specification language, which in certain situations (see DC) results in a more efficient modelling.

5 Final remarks

We have proposed, implemented, and experimentally evaluated SAT- and BDD-based bounded model checking approaches for ELTLK interpreted over both the standard interpreted systems and the interleaved interpreted systems. The experimental results show that the approaches are complementary, and that the BDD-based BMC approach appears to be superior for the IIS semantics, while the SAT-based approach appears to be superior for the IS semantics. This is a novel and interesting result, which shows that the choice of the semantics should depend on the symbolic method applied.

We have also done our best to provide a comparison of our BMC methods with the MCK tool. This comparison shows that the efficiency of the verification approach is strongly influenced by the semantics used to model MAS, i.e., whether IS or IIS are applied.

In the future we are going to extend the presented algorithms to handle also the ECTL*K properties.

Acknowledgments Partly supported by National Science Center under the Grant No. 2011/01/B/ST6/05317 and 2011/01/B/ST6/01477. Artur Męski acknowledges the support of the EU, European Social Fund. Project PO KL "Information technologies: Research and their interdisciplinary applications" (UDA-POKL.04.01.01-00-051/10-00).

References

1. Abdulla, P. A., Bjesse, P., & Eén, N. (2000). Symbolic reachability analysis based on SAT-solvers. In *Proceedings of the 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'00). Lecture Notes in Computer Science*, (Vol. 1785, pp. 411–425). Berlin: Springer.
2. Biere, A. (2008). PicoSAT essentials. *Journal on Satisfiability Boolean Modeling and Computation (JSAT)*, *4*, 75–97.
3. Biere, A., Cimatti, A., Clarke, E., Fujita, M., & Zhu, Y. (1999). Symbolic model checking using SAT procedures instead of BDDs. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC'99)* (pp. 317–320).
4. A. Biere, A. Cimatti, E. Clarke, & Y. Zhu. (1999). Symbolic model checking without BDDs. In *Proceedings of the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99). Lecture Notes in Computer Science* (Vol. 1579, pp. 193–207). Berlin: Springer.
5. Biere, A., Heljanko, K., Junttila, T., Latvala, T., & Schuppan, V. (2006). Linear encodings of bounded LTL model checking. *Logical Methods in Computer Science*, *2*(5:5), 1–64.
6. R. Bordini, M. Fisher, C. Pardavila, W. Visser, & M. Wooldridge. (2003). Model checking multi-agent programs with CASP. In *Proceedings of the 15th International Conference on Computer Aided Verification (CAV'03). Lecture Notes in Computer Science* (Vol. 2725, pp. 110–113). Springer.
7. Bordini, R. H., Fisher, M., Wooldridge, M., & Visser, W. (2009). Property-based slicing for agent verification. *Journal of Logic and Computation*, *19*(6), 1385–1425.
8. N. Bulling & W. Jamroga. (2010). Model checking agents with memory is harder than it seemed. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'10)* (pp. 633–640). International Foundation for Autonomous Agents and Multiagent Systems.
9. Cabodi, G., Camurati, P., & Quer, S. (2002). Can BDD compete with SAT solvers on bounded model checking?. In *Proceedings of the 39th Design Automation Conference (DAC'02)* (pp. 117–122).
10. Chaum, D. (1988). The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, *1*(1), 65–75.
11. Clarke, E., Grumberg, O., & Hamaguchi, K. (1994). Another look at LTL model checking. In *Proceedings of the 6th International Conference on Computer Aided Verification (CAV'94). Lecture Notes in Computer Science* (Vol. 818, pp. 415–427). Berlin: Springer.
12. Clarke, E., Grumberg, O., & Peled, D. (1999). *Model checking*. Cambridge: MIT Press.
13. Copty, F., Fix, L., Fraer, R.., Giunchiglia, E., Kamhi, G.., Tacchella, A., & Vardi, M. (2001). Benefits of bounded model checking at an industrial setting. In *Proceedings of the 13th International Conference on Computer Aided Verification (CAV'01). Lecture Notes in Computer Science* (Vol. 2102, pp. 436–453). Berlin: Springer.
14. Dennis, L. A., Fisher, M., Webster, M. P., & Bordini, R. H. (2012). Model checking agent programming languages. *Automated Software Engineering*, *19*(1), 5–63.
15. Etessami, K., & Holzmann, G. J. (2000). Optimizing büchi automata. In *Proceedings of the 11th International Conference on Concurrency Theory (CONCUR'00). Lecture Notes in Computer Science* (Vol. 1877, pp. 153–167). Berlin: Springer.
16. Fagin, R., Halpern, J. Y., Moses, Y., & Vardi, M. (1995). *Reasoning about Knowledge*. Cambridge: MIT Press.
17. Gammie, P., & Meyden, R. (2004). MCK: Model checking the logic of knowledge. In *Proceedings of the 16th International Conference on Computer Aided Verification (CAV'04). Lecture Notes in Computer Science* (Vol. 3114, pp. 479–483). Berlin: Springer.

18. Gastin, P., & Oddoux, D. (2001). Fast LTL to Büchi automata translation. In *Proceedings of the 13th International Conference on Computer Aided Verification (CAV'01). Lecture Notes in Computer Science* (Vol. 2102, pp. 53–65). Berlin: Springer.
19. Gerth, R., Peled, D., Vardi, M., & Wolper, P. (1995). Simple on-the-fly automatic verification of linear temporal logic. In *Proceedings of IFIP/WG6.1 Symposium. Protocol Specification, Testing and Verification (PSTV'95)* (pp. 3–18). Chapman & Hall.
20. Halpern, J., & Vardi, M. (1991). Model checking vs. theorem proving: A manifesto. In *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)* (pp. 325–334). Cambridge: Morgan Kaufmann.
21. Hoek, W., & Wooldridge, M. (2003). Cooperation, knowledge, and time: Alternating-time temporal epistemic logic and its applications. *Studia Logica*, *75*(1), 125–157.
22. Hoek, W. V., & Wooldridge, M. (2002). Model checking knowledge and time. In *Proceedings of the 9th International SPIN Workshop on Model Checking of Software (SPIN'2002). Lecture Notes in Computer Science* (Vol. 2318, pp. 95–111). Berlin: Springer.
23. Huang, X., Luo, C., & van der Meyden, R. (2011). Improved bounded model checking for a fair branching-time temporal epistemic logic. In *Proceedings of the 6th Workshop on Model Checking and Artificial Intelligence (MoChArt'2010), LNAI* (Vol. 6572, pp. 95–111). Berlin: Springer.
24. Jamroga, W., & Dix, J. (2008). Model checking abilities of agents: A closer look. *Theory of Computing Systems*, *42*(3), 366–410.
25. Jamroga, W., & Penczek, W. (2012). Specification and verification of multi-agent systems. In *Lectures on Logic and Computation (ESSLLI'2010, ESSLLI'2011). Lecture Notes in Computer Science* (Vol. 7388, pp. 210–263). Berlin: Springer.
26. Jones, A. V., & Lomuscio, A. (2010). Distributed BDD-based BMC for the verification of multi-agent systems. In *Proceedings of the 9th International Conference on Autonomous Agents and Multi-Agent systems (AAMAS'2010)* (pp. 675–682). Toronto: IFAAMAS Press.
27. Kacprzak, M., Lomuscio, A., Niewiadomski, A., Penczek, W., Raimondi, F., & Szreter, M. (2006). Comparing BDD and SAT based techniques for model checking Chaum's dining cryptographers protocol. *Fundamenta Informaticae*, *72*(1–2), 215–234.
28. Kacprzak, M., Nabiałek, W., Niewiadomski, A., Penczek, W., Półrola, A., Szreter, M., et al. (2008). Verics 2007—a model checker for knowledge and real-time. *Fundamenta Informaticae*, *85*(1–4), 313–328.
29. Lomuscio, A., Lasica, T., & Penczek, W. (2003). Bounded model checking for interpreted systems: Preliminary experimental results. In *Proceedings of the 2nd NASA Workshop on Formal Approaches to Agent-Based Systems (FAABS'02), LNAI* (Vol. 2699, pp. 115–125). Berlin: Springer.
30. Lomuscio, A., Pecheur, C., & Raimondi, F. (2007). Automatic verification of knowledge and time with nusmv. In *Proceedings of International Conference on Artificial Intelligence (IJCAI'07)* (pp. 1384–1389).
31. Lomuscio, A., Penczek, W., & Qu, H. (2010). Partial order reduction for model checking interleaved multi-agent systems. In *Proceedings of the 9th International Conference on Autonomous Agents and Multi-Agent systems (AAMAS'2010)* (pp. 659–666). Toronto: FAAMAS Press.
32. Lomuscio, A., Penczek, W., & Woźna, B. (2007). Bounded model checking for knowledge and real time. *Artificial Intelligence*, *171*, 1011–1038.
33. Męski, A., Penczek, W., & Szreter, M. (2011). Bounded model checking linear time and knowledge using decision diagrams. In *Proceedings of the International Workshop on Concurrency, Specification and Programming (CS&P'11)* (pp. 363–375).
34. Męski, A., Penczek, W., & Szreter, M. (2012). BDD-based bounded model checking for LTLK over two variants of interpreted systems. In *Proceedings of 5th International Workshop on Logics, Agents, and Mobility* (pp. 35–50).
35. Męski, A., Penczek, W., Szreter, M., Woźna-Szcześniak, B., & Zbrzezny, A. (2012). Bounded model checking for knowledge and linear time. In *Proceedings of the 11th International Conference on Autonomous Agents and Multi-Agent systems (AAMAS'2012)* (pp. 1447–1448). Toronto: IFAAMAS Press.
36. Męski, A., Penczek, W., Szreter, M., Woźna-Szcześniak, B., & Zbrzezny, A. (2012). Two approaches to bounded model checking for linear time logic with knowledge. In *The Proceedings of the 6th KES International Conference on Agent and Multi-Agent Systems, Technologies and Applications (KES-AMSTA'2012). Lecture Notes in Computer Science* (Vol. 7327, pp. 514–523). Berlin: Springer.
37. Męski, A., Woźna-Szcześniak, B., Zbrzezny, A. M., & Zbrzezny, A. (2013). Two approaches to bounded model checking for a soft real-time epistemic computation tree logic. In *Proceedings of the 10th International Symposium on Distributed Computing and Artificial Intelligence (DCAI'2013), Advances in Intelligent and Soft-Computing*, (Vol. 217, pp. 483–492). Berlin: Springer.
38. Meyden, R., & Shilov, N. V. (1999). Model checking knowledge and time in systems with perfect recall. In *Proceedings of the 19th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'99). Lecture Notes in Computer Science* (Vol. 1738, pp. 432–445). Berlin: Springer.

39. Meyden, R., & Su, K. (2004). Symbolic model checking the knowledge of the dining cryptographers. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop (CSFW-17)* (pp. 280–291). IEEE Computer Society.
40. Peled D. (1993). All from one, one for all: On model checking using representatives. in *Proceedings of the 5th International Conference on Computer Aided Verification (CAV'93). Lecture Notes in Computer Science* (Vol. 697, pp. 409–423). Berlin: Springer.
41. Penczek, W., & Lomuscio, A. (2003). Verifying epistemic properties of multi-agent systems via bounded model checking. *Fundamenta Informaticae*, *55*(2), 167–185.
42. Penczek, W., Woźna-Szcześniak, B., & Zbrzezny, A. (2012). Towards SAT-based BMC for LTLK over interleaved interpreted systems. *Fundamenta Informaticae*, *119*(3–4), 373–392.
43. Raimondi, F., & Lomuscio, A. (2007). Automatic verification of multi-agent systems by model checking via OBDDs. *Journal of Applied Logic*, *5*(2), 235–251.
44. Somenzi, F. CUDD: CU decision diagram package—release 2.3.1. http://vlsi.colorado.edu/~fabio/CUDD/cuddIntro.html.
45. Somenzi, F., Bloem, R. (2000). Efficient Büchi automata from LTL formulae. In *Proceedings of the 12th International Conference on Computer Aided Verification (CAV'00). Lecture Notes in Computer Science* (Vol. 1855, pp. 248–263). Berlin: Springer.
46. Su, K., Sattar, A., & Luo, X. (2007). Model checking temporal logics of knowledge via OBDDs. *The Computer Journal*, *50*(4), 403–420.
47. Troquard, N., Hoek, W. V. D., & Wooldridge, M. (2009). Model checking strategic equilibria. In *Proceedings of the 5th International Workshop on Model Checking and Artificial Intelligence (MOCHART'2008), LNAI* (Vol. 5348, pp. 166–188). Berlin: Springer.
48. Wooldridge, M. (2002). *An introduction to multiagent systems*. Chichester: Wiley.
49. Woźna, B., Lomuscio, A., & Penczek, W. (2005). Bounded model checking for deontic interpreted systems. In *Proceedings of the 2nd International Workshop on Logic and Communication in Multi-Agent Systems (LCMAS'04), ENTCS* (Vol. 126, pp. 93–114). Amsterdam: Elsevier.
50. Woźna, B., Zbrzezny, A., & Penczek, W. (2003). Checking reachability properties for timed automata via SAT. *Fundamenta Informaticae*, *55*(2), 223–241.
51. Woźna-Szcześniak, B., & Zbrzezny, A. (2012). Sat-based bounded model checking for deontic interleaved interpreted systems. In *The Proceedings of the 6th KES International Conference on Agent and Multi-Agent Systems, Technologies and Applications (KES-AMSTA'2012). Lecture Notes in Computer Science* (Vol. 7327, pp. 494–503). Berlin: Springer.
52. Woźna-Szcześniak, B., & Zbrzezny, A. (2013). SAT-based bmc for deontic metric temporal logic and deontic interleaved interpreted systems. In *Declarative Agent Languages and Technologies X. The 10th International Workshop (DALT'2012), LNAI* (Vol. 7784, pp. 70–189). Berlin: Springer.
53. Woźna-Szcześniak, B., Zbrzezny, A. M., & Zbrzezny, A. (2011). The BMC method for the existential part of RTCTLK and interleaved interpreted systems. In *In Proceedings of the 15th Portuguese Conference on Artificial Intelligence (EPIA'2011), LNAI* (Vol. 7026, pp. 551–565). Berlin: Springer.
54. Zbrzezny, A. (2008). Improving the translation from ECTL to SAT. *Fundamenta Informaticae*, *85*(1–4), 513–531.
55. Zbrzezny, A. (2012). A new translation from ECTL* to SAT. *Fundamenta Informaticae*, *120*(3–4), 377–397.