Fundamenta Informaticae 59 (2007) 1–15
IOS Press

VerICS 2007 - a Model Checker for Knowledge and Real-Time[*]

Magdalena Kacprzak
Bialystok University of Technology, FCS, Wiejska 45a, 15-351 Bialystok, Poland

Wojciech Nabiałek, Artur Niewiadomski
Podlasie Academy, ICS, Sienkiewicza 51, 08-110 Siedlce, Poland

Wojciech Penczek[†]
ICS PAS, Ordona 21, 01-237 Warsaw, Poland
Podlasie Academy, ICS, Sienkiewicza 51, 08-110 Siedlce, Poland

Agata Półrola
University of Lodz, FMCS, Banacha 22, 90-238 Lodz, Poland

Maciej Szreter
ICS PAS, Ordona 21, 01-237 Warsaw, Poland

Bożena Woźna, Andrzej Zbrzezny
Jan Dlugosz Academy, IMCS, Armii Krajowej 13/15, 42-200 Czestochowa, Poland

Abstract. The papers presents the current stage of the development of VerICS - a model checker for real-time and multi-agent systems. Depending on the type of a system considered, it enables to test various classes of properties - from reachability to temporal, epistemic and deontic formulas. The model checking methods used to this aim include both SAT-based and enumerative ones. In the paper we focus on new features of the verifier: SAT-based model checking for multi-agent systems and several extensions and improvements to real-time systems' verification.

[*]Partly supported by the Ministry of Education and Science under the grant No. 3 T11C 011 28
[†]Address for correspondence: The VerICS Group, ICS PAS, Ordona 21, 01-237 Warsaw, Poland; `verics@ipipan.waw.pl`

1. Introduction

The paper presents the current stage of the development of VerICS, a model checker for real-time and multi-agent systems. Depending on the type of a system considered, the verifier enables to test various classes of properties - from reachability of a state satisfying certain conditions to more complicated features expressed by formulas of (timed) temporal, epistemic, or deontic logics. The model checking methods implemented include both SAT-based and enumerative ones (where by the latter we mean these consisting in generating abstract models for systems). Our first work [3] presenting VerICS dealt mainly with verification of Real-Time Systems (RTS). In this paper we focus on VerICS' new features, i.e., SAT-based model checking for Multi-Agent Systems (MAS), and several extensions to RTS verification.

A survey of model checkers for RTS can be found in [17]. Considering MAS verification, VerICS is, to our best knowledge, one of the three existing model checkers for verifying MAS directly, and the only one which applies SAT to this aim. The other two: MCMAS [12] and MCK [6] implement BDD-based verification methods. Some other tools like CASP [1, 21] or MABLE [21] enable translations from MAS to languages accepted by "general purpose" model checkers like Spin or JavaPathFinder.

The rest of the paper is organised as follows. In Sec. 2 we briefly present the theoretical background of the SAT-based verification methods implemented in our tool (i.e., bounded and unbounded model checking). The next two sections contain a description of the system: Sec. 3 presents the architecture of VerICS, whereas Sec. 4 gives an overview of the tool. Next, in Sec. 5 we provide some experimental results obtained for several typical benchmarks used to test efficiency of model checkers. Finally, Sec. 6 contains a summary and some concluding remarks.

2. Theoretical Background

A network of communicating (timed) automata is the basic VerICS' formalism for modelling a system to be verified. Timed automata are used to specify RTS (possibly with clock differences expressing constraints on their behaviour), whereas timed or untimed automata are applied to model MAS (possibly extended in a way to handle certain features of interest, like deontic automata in [9]).

The tuples of *local states* of the automata in a network N define the *global states* of the system considered. The set of all the possible *runs* (i.e., infinite evolutions from a given initial state) of an RTS modelled by N gives us a *computation tree* which, after labelling the states with propositions from a given set PV which are true at these states (i.e., changing the tree into a *model*), is used to interpret the formulas of timed or untimed temporal logics (like CTL or TCTL) expressing properties to be checked. In the case of modelling a MAS we augment the model with *epistemic* or *deontic accessibility relations*. The resulting structure enables us to interpret formulas involving temporal operators, epistemic operators - to reason about *knowledge* of agents [5], and deontic operators - to reason about *correctness* of their behaviour.

SAT-based verification methods represent the models and properties of systems in the form of boolean formulas in order to reduce the state explosion. These for MAS involve bounded (BMC) and unbounded model checking (UMC). Currently, VerICS implements UMC for $\mathrm{CTL_pK}$ (Computation Tree Logic with knowledge and past operators) [8], and BMC for ECTLKD (the existential fragment of CTL extended with knowledge and deontic operators) [9, 16, 22, 23] as well as TECTLK (the existential fragment of timed CTL extended with knowledge operators) [13]. Considering verification of RTS, the current version of VerICS offers BMC for proving (un)reachability [24] (also for timed automata with clock differences [25]), and UMC for proving CTL properties for slightly restricted timed automata [19]. Below we present some more details of BMC for ECTLKD and UMC for $\mathrm{CTL_pK}$. Verification of TECTLK formulas is performed by reducing TECTLK model checking problem to ECTLK model checking problem, and following BMC procedure shown below (see [11] for details).

2.1. Bounded Model Checking

Bounded Model Checking (BMC) is a symbolic method aimed at verification of temporal properties of distributed (timed) systems. It is based on the observation that some properties of a system can be checked over a part of its model only. In the simplest case of reachability analysis, this approach consists in an iterative encoding of a finite symbolic path (computation) as a propositional formula. The satisfiability of the resulting propositional formula is then checked using an external SAT-solver.

Consider a system consisting of n agents such that the local states of each agent are divided into allowed or disallowed. The sets of allowed (*green*) and disallowed (*red*) states of the agent i, denoted respectively $\mathcal{G}_i$ and $\mathcal{R}_i$, are disjoint; moreover, the set $\mathcal{G}_i$ is nonempty. Let G be the set of global states (i.e., tuples of n local states) of the system under consideration, g^0 be its initial state, $Q \subseteq G$ be the set of states reachable from g^0, $T \subseteq G \times G$ be a transition relation, $\sim_i, \sim_i^O \subseteq G \times G$ (for $i = 1, \ldots, n$), be, respectively, the epistemic and deontic accessibility relation[1], and $V : G \to 2^{PV}$ be a valuation function. Moreover, let for $k \in \mathbb{N}_+$ a *k-path* be a finite sequence of $k+1$ states $\pi = (g_0, \ldots, g_k)$, where $g_i \in G$ for $i = 0, \ldots, k$ and $(g_i, g_{i+1}) \in T$ for each $0 \le i < k$. For a k-path $\pi = (g_0, \ldots, g_k)$, let $\pi(i) = g_i$ for each $0 \le i \le k$. By $\Pi_k(g)$ we denote the set of all the k-paths starting at g. We assume that the reader is familiar with the standard syntax and semantics of CTL, CTLK, CTLKD, and $\mathrm{CTL_pK}$ (details can be found in [10]). In order to restrict the semantics of ECTLKD to a part of the model we introduce the following definition:

Definition 2.1. Let $M = (G, Q, g^0, T, \sim_1, \ldots, \sim_n, \sim_1^O, \ldots, \sim_n^O, V)$ be a model and $k \in \mathbb{N}_+$. The $k-$*model* for M is defined as $M_k = (Q, g^0, P_k, \sim_1, \ldots, \sim_n, \sim_1^O, \ldots, \sim_n^O, V')$, where P_k is the set of all the k-paths of M over Q, i.e., $P_k = \bigcup_{s \in Q} \Pi_k(s)$, and $V' = V|_Q$.

In order to identify k-paths that represent infinite paths we define the function $loop : P_k \to 2^{\mathbb{N}}$ as: $loop(\pi) = \{l \mid 0 \le l \le k \ and \ (\pi(k), \ \pi(l)) \in T\}$, which returns the set of indices l of π for which there is a transition from $\pi(k)$ to $\pi(l)$. Then, we define a *bounded semantics* of ECTLKD (we omit here the operators $\overline{\mathrm{D}}$, $\overline{\mathrm{E}}$ and $\overline{\mathrm{C}}$):

Definition 2.2. Let M_k be a $k-$model and α, β be ECTLKD formulas. $M_k, s \models \alpha$ denotes that α is true at the state s of M_k. M_k is omitted if it is clear from the context. The relation $\models$ for modal operators is defined inductively as follows:

[1]Two global states g, g' are in the relation $\sim_i$ if they share the same i-th local states, and are in the relation $\sim_i^O$ if the i-th local state of g' is green (i.e., it is in $\mathcal{G}_i$)

$s \models \mathrm{EX}\alpha$	iff	$(\exists \pi \in P_k(s))\ \pi(1) \models \alpha$,
$s \models \mathrm{EG}\alpha$	iff	$(\exists \pi \in P_k(s))(\forall 0 \leq j \leq k)(\pi(j) \models \alpha$ and $loop(\pi) \neq \emptyset)$,
$s \models \mathrm{E}(\alpha \mathrm{U} \beta)$	iff	$(\exists \pi \in P_k(s))(\exists 0 \leq j \leq k)\big(\pi(j) \models \beta$ and $(\forall 0 \leq i < j)\pi(i) \models \alpha\big)$,
$s \models \overline{\mathrm{K}}_i\alpha$	iff	$(\exists \pi \in P_k(\iota))(\exists 0 \leq j \leq k)\big(\pi(j) \models \alpha$ and $s \sim_i \pi(j)\big)$,
$s \models \overline{\mathrm{K}}_i^l\alpha$	iff	$(\exists \pi \in P_k(\iota))(\exists 0 \leq j \leq k)\big(\pi(j) \models \alpha$ and $s \sim_i \pi(j)$ and $s \sim_l^O \pi(j)\big)$.

Model checking over models can be reduced to model checking over k-models. The main idea of BMC for ECTLKD is that we can check φ over M_k by testing the satisfiability of the propositional formula $[M, \varphi]_k := [M^{\varphi, g^0}]_k \wedge [\varphi]_{M_k}$, where the first conjunct represents (a part of) the model under consideration and the second a number of constraints that must be satisfied on M_k for φ to be satisfied. Once this translation is defined, checking satisfiability of an ECTLKD formula can be done by means of a SAT-checker. Typically, we start with $k := 1$, test satisfiability for the translation, and increase k by one until either $[M^{\varphi, g^0}]_k \wedge [\varphi]_{M_k}$ becomes satisfiable, or k reaches the maximal depth of M.[2]

2.1.1. Translation

Below, we provide some details of the translation. Given a MAS represented by a network of automata. Each global state g of the system can be represented by $w = (w[1], \ldots, w[m])$, for some $m \in \mathbb{N}_+$, (which we shall call *a global state variable*), where each $w[i]$ for $i = 1, \ldots, m$ is a propositional variable. A sequence $w_{0,j}, \ldots, w_{k,j}$ of global state variables is called *a symbolic k-path j*.

Let f_k be a function which determines the number of k-paths sufficient for checking a given ECTLKD formula (see [22]). Moreover, let $\overline{PV} = \{\neg p \mid p \in PV\}$ be the set of negated propositions of PV, and let *lit*: $\{0, 1\} \times PV \to PV \cup \overline{PV}$ be a function defined as follows: $lit(0, p) = \neg p$ and $lit(1, p) = p$. Furthermore, let w, v be two global state variables. We use the formula $I_g(w) := \bigwedge_{i=1}^{m} lit(g_i, w[i])$ to encode a global state $g = (g_1, \ldots, g_m)$ of the model, i.e., if $g_i = 1$, then it is encoded by $w[i]$, and if $g_i = 0$, then it is encoded by $\neg w[i]$. The propositional formula $[M^{\varphi, g^0}]_k$, representing the k-paths in the k-model, is defined as $[M^{\varphi, g^0}]_k := I_{g^0}(w_{0,0}) \wedge \bigwedge_{j=1}^{f_k(\varphi)} \bigwedge_{i=0}^{k-1} T(w_{i,j}, w_{i+1,j})$, where $w_{0,0}$ and $w_{i,j}$ for $0 \leq i \leq k$ and $1 \leq j \leq f_k(\varphi)$ are global state variables, and $T(w_{i,j}, w_{i+1,j})$ is a formula encoding the transition relation T.

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

- $p(w)$ encodes a proposition p of ECTLKD over w;
- $H(w, v) := \bigwedge_{j=1}^{m} w[j] \Leftrightarrow v[j]$ represents logical equivalence between global state encodings u and v (i.e., encodes that u and v represent the same global states);
- $HK_i(w, v) := \bigwedge_{j \in Ix_i} w[j] \Leftrightarrow v[j]$ represents logical equivalence between i-local state encodings u and v, (i.e., encodes that u and v share i-local states);
- $HP_i(w, v)$ encodes the set of all global states in which agent i is running correctly;

[2]The upper limit is $|Q|$.

- $L_{k,j}(l)$ encodes a backward loop connecting the k-th state to the l-th state in the symbolic $k-$path j, for $0 \leq l \leq k$.

The translation of φ at the state $w_{m,n}$ into the propositional formula $[\varphi]_k^{[m,n]}$ is as follows (we give the translation of selected formulas only):

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

Given the translations above, we can now check φ over M_k by checking the satisfiability of the propositional formula $[M^{\varphi,g^0}]_k \wedge [\varphi]_{M_k}$, where $[\varphi]_{M_k} = [\varphi]_k^{[0,0]}$.

2.2. Unbounded Model Checking

Unlike BMC, UMC is capable of handling the whole language of the logic. Our aim is to translate $\mathrm{CTL_pK}$ formulas into propositional formulas in conjunctive normal form. Specifically, for a given $\mathrm{CTL_pK}$ formula φ we compute a corresponding propositional formula $[\varphi](w)$, where w is a global state variable (i.e., a vector of propositional variables for representing global states) encoding these states of the model where φ holds. To calculate the actual translations we use either the QBF or the fixed-point characterisation of $\mathrm{CTL_pK}$ formulas. Below, we recall the necessary notions.

2.2.1. Formulas in CNF and QBF

Let PV be a finite set of propositional variables. A *literal* is a propositional variable $p \in PV$ or its negation $\neg p$. A *clause* is a disjunction of a set of zero or more literals $l[1] \vee \ldots \vee l[n]$. A disjunction of zero literals is taken to mean the constant **false**. A formula is in a *conjunctive normal form* (CNF) if it is a conjunction of a set of zero or more clauses $c[1] \wedge \ldots \wedge c[n]$. An *assignment* A is a partial function from PV to $\{\mathbf{true}, \mathbf{false}\}$.

In our method, in order to have a more succinct notation for complex operations on boolean formulas, we also use *Quantified Boolean Formulas* (QBF), an extension of propositional logic by means of quantifiers ranging over propositions. The BNF syntax of a QBF formula is given by:

$$\alpha ::= p \mid \neg\alpha \mid \alpha \wedge \alpha \mid \exists p.\alpha \mid \forall p.\alpha.$$

The semantics of the quantifiers is defined by $\exists p.\alpha$ iff $\alpha(p \leftarrow \mathbf{true}) \vee \alpha(p \leftarrow \mathbf{false})$, and $\forall p.\alpha$ iff $\alpha(p \leftarrow \mathbf{true}) \wedge \alpha(p \leftarrow \mathbf{false})$, where $\alpha \in$ QBF, $p \in PV$ and $\alpha(p \leftarrow q)$ denotes substitution with the variable q of every occurrence of the variable p in formula α. We will use the notation $\forall v.\alpha$, where $v = (v[1], \ldots, v[m])$ is a vector of propositional variables, to denote $\forall v[1].\forall v[2] \ldots \forall v[m].\alpha$. For a given QBF formula $\forall v.\alpha$, we can construct a CNF formula equivalent to it by using the algorithm *forall* [14].

Our aim is to translate the whole $\mathrm{CTL_pK}$ language into boolean formulas. To this aim, we first need to translate $\mathrm{CTL_pK}$ formulas into QBF form. Before doing so, we need to be able to present the $\mathrm{CTL_pK}$ language in terms of fixed-points. This is shown in the next subsection. In our presentation, we follow and adapt definitions given in [2].

2.2.2. Fixed-point characterisation of $\mathrm{CTL_pK}$

Let $M = (G, Q, g^0, T, \sim_1, \ldots, \sim_n, \sim_1^O, \ldots, \sim_n^O, V)$ be a model defined as in Sec. 2.1. Notice that the set 2^G of all subsets of G forms a lattice under the set inclusion ordering. Each element G' of the lattice can also be thought of as a *predicate* on G, where the predicate is viewed as being true for exactly the states in G'. The least element in the lattice is the empty set, which corresponds to the predicate **false**, and the greatest element in the lattice is the set G, which corresponds to **true**. A function τ mapping 2^G to 2^G is called a *predicate transformer*. A set $G' \subseteq G$ is a *fixed point* of a function $\tau : 2^G \to 2^G$ if $\tau(G') = G'$.

Whenever τ is monotonic (i.e., when $P \subseteq P'$ implies $\tau(P) \subseteq \tau(P')$), a function τ has a least fixed point denoted by $\mu Z.\tau(Z)$, and a greatest fixed point, denoted by $\nu Z.\tau(Z)$. When τ is monotonic and $\bigcup$-continuous (i.e., when $P_1 \subseteq P_2 \subseteq \ldots$ implies $\tau(\bigcup_i P_i) = \bigcup_i \tau(P_i)$), then $\mu Z.\tau(Z) = \bigcup_{i\geq 0} \tau^i(\textbf{false})$. When τ is monotonic and $\bigcap$-continuous (i.e., when $P_1 \supseteq P_2 \supseteq \ldots$ implies $\tau(\bigcap_i P_i) = \bigcap_i \tau(P_i)$), then $\nu Z.\tau(Z) = \bigcap_{i\geq 0} \tau^i(\textbf{true})$ (see [20]).

In order to obtain fixed-point characterisations of the modal operators, we identify each $\mathrm{CTL_pK}$ formula α with the set $\langle\alpha\rangle_{\mathrm{M}}$ of the states in M at which this formula is true, formally $\langle\alpha\rangle_{\mathrm{M}} = \{s \in G \mid M, s \models \alpha\}$. If M is clear from the context we omit the subscript M. Furthermore, we define functions $\mathrm{AX}, \mathrm{AY}, \mathrm{K}_i$ from 2^G to 2^G as follows:

- $\mathrm{AX}(Z) = \{g \in G \mid \forall g' \in G \text{ if } (g, g') \in T, \text{ then } s' \in Z\}$,
- $\mathrm{AY}(Z) = \{g \in G \mid \forall g' \in G \text{ if } (g', g) \in T, \text{ then } s' \in Z\}$,
- $\mathrm{K}_i(Z) = \{g \in G \mid \forall g' \in G \text{ if } (g^0, g') \in T^* \text{ and } g \sim g', \text{ then } g' \in Z\}$,

Observe that $\langle O\alpha\rangle = O(\langle\alpha\rangle)$, for $O \in \{\mathrm{AX}, \mathrm{AY}, \mathrm{K}_i\}$. Then, the following temporal and epistemic operators may be characterised as the least or the greatest fixed point of an appropriate monotonic ($\bigcap$-continuous or $\bigcup$-continuous) predicate transformer (see [4, 2]): $\langle \mathrm{AG}\alpha\rangle = \nu Z.\langle\alpha\rangle \cap \mathrm{AX}(Z)$, $\langle \mathrm{A}(\alpha \mathrm{U} \beta)\rangle = \mu Z.\langle\beta\rangle \cup (\langle\alpha\rangle \cap \mathrm{AX}(Z))$, $\langle \mathrm{AH}\alpha\rangle = \nu Z.\langle\alpha\rangle \cap \mathrm{AY}(Z)$.

2.2.3. Unbounded Model Checking on $\mathrm{CTL_pK}$

Given a model M, we encode its states like for BMC in Sec. 2.1.1. Now, our aim is to translate $\mathrm{CTL_pK}$ formulas into propositional formulas. Specifically, for a given $\mathrm{CTL_pK}$ formula β we compute a corresponding propositional formula $[\beta](w)$, which encodes those states of the system that satisfy the formula. Operationally, we work outwards from the most nested subformulas, i.e., the atoms. In other words, to compute $[\mathrm{O}\alpha](w)$, where O is a modality, we work under the assumption of already having computed $[\alpha](w)$. To calculate the actual translations we use either the fixed-point or the QBF characterisation of $\mathrm{CTL_pK}$ formulas. For example, the formula $[\mathrm{AX}\alpha](w)$ is equivalent to the QBF formula $\forall v.(T(w, v) \Rightarrow [\alpha](v))$. We can use similar equivalences for formulas $\mathrm{AY}\alpha, \mathrm{K}_i\alpha$. More specifically, we use three basic algorithms: The first one, implemented by the procedure *forall* [14], is used for formulas

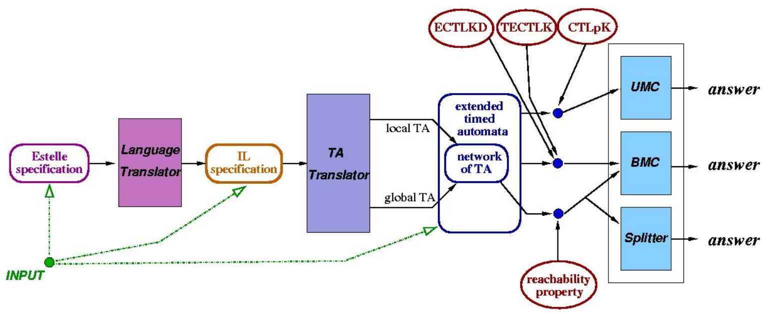


Figure 1. Architecture of VerICS

$\mathrm{O}\alpha$ such that $\mathrm{O} \in \{\mathrm{AX}, \mathrm{AY}, \mathrm{K}_i\}$. This procedure eliminates the universal quantifier from a QBF formula representing a $\mathrm{CTL_pK}$ formula, and returns the result in a conjunctive normal form. The second algorithm [8], implemented by the procedure *gfp*$_O$, is applied to formulas $\mathrm{O}\alpha$ such that $\mathrm{O} \in \{\mathrm{AG}, \mathrm{AH}\}$. This procedure computes the greatest fixed point. For formulas of the form $\mathrm{A}(\alpha \mathrm{U} \beta)$ we use the third procedure, called *lfp*$_{AU}$, which computes the least fixed point. In so doing, given a formula β we obtain a propositional formula $[\beta](w)$ such that β is valid in the model M iff the propositional formula $[\beta](w) \wedge I_{g^0}(w)$ is satisfiable.

3. Implementation

The architecture of VerICS is shown in Fig. 1. The system consists of:

- **Estelle to Intermediate Language (IL) translator**, which enables to handle specifications written in a subset of Estelle [7] (the standardised language for specifying communicating protocols and distributed systems);
- **IL to timed automata translator**, which, given an IL specification, generates the corresponding network of timed automata or the global timed automaton;
- **BMC module**, which implements BMC-based verification for the classes of properties shown in the figure. The SAT-solver used is MiniSat [15] or RSat [18]; the system can be configured to work with other solvers;
- **UMC module**, which provides preliminary implementations of UMC verification methods for properties described above. The module is integrated with a modified version of the SAT-solver ZChaff [26];
- **Splitter module**, which performs reachability verification on abstract models generated for timed automata.

VerICS has been implemented in C++; its internal functionalities are available via a interface written in Java. The current distribution (binaries of a standalone program to be run under Linux, or a client version which can work under an arbitrary Java-supporting operating system) can be downloaded from `http://verics.ipipan.waw.pl`. A more detailed description of the tool is presented in the following section.

4. VerICS - an Overview

Our model checker can be downloaded from the web page of the address `http://verics.ipipan.waw.pl`. The page offers two kinds of binaries: the full (standalone) version of VerICS to be run under Linux (the additional requirement is the Java runtime environment (JRE) version 1.5 or later), and the graphical interface of the client for any Java-supporting operating system (in this case, JRE version 1.5 or later, and the internet connection on the port 8080 are required). It can be run by the file `run.bat` or `run.sh`, depending on the operating system. The starting screen presents the architecture of the verifier (see Fig. 2).

In order to input a system to be tested we can use VerICS' graphical editor which enables drawing timed/deontic automata (Fig. 3). Each automaton of a network we consider should be given on

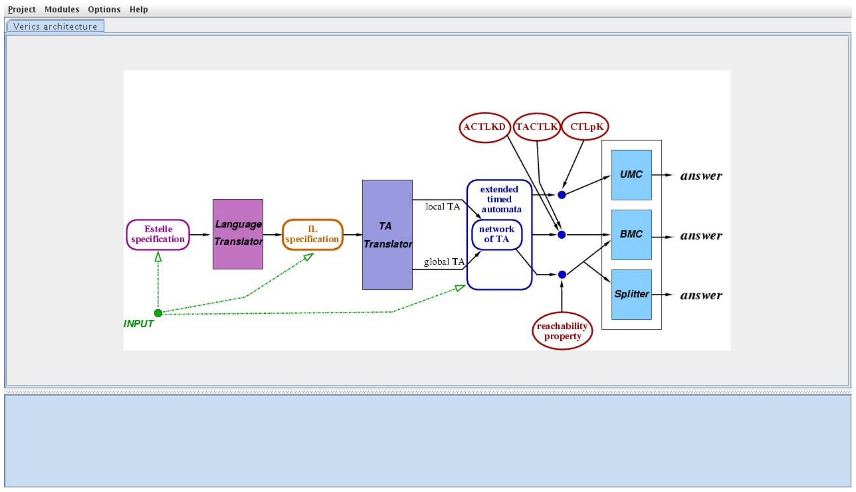
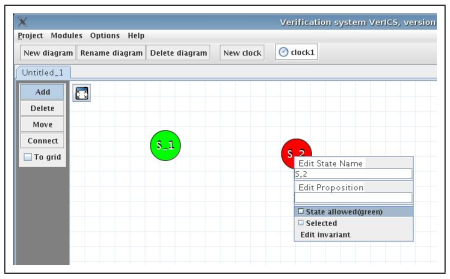


Figure 2. VerICS' graphical interface and the start screen



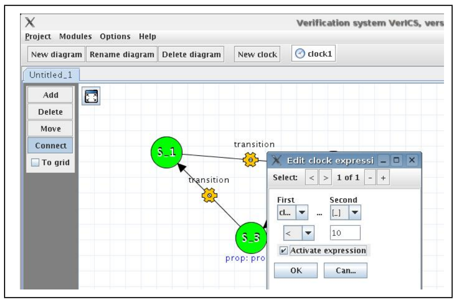
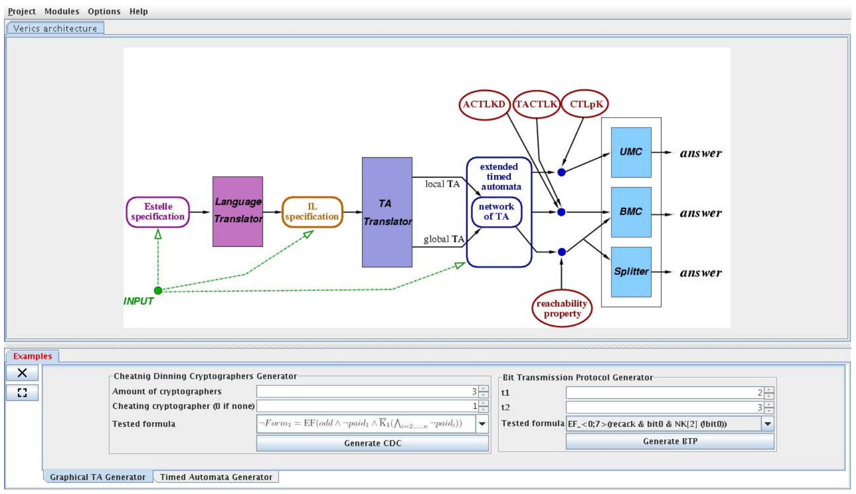


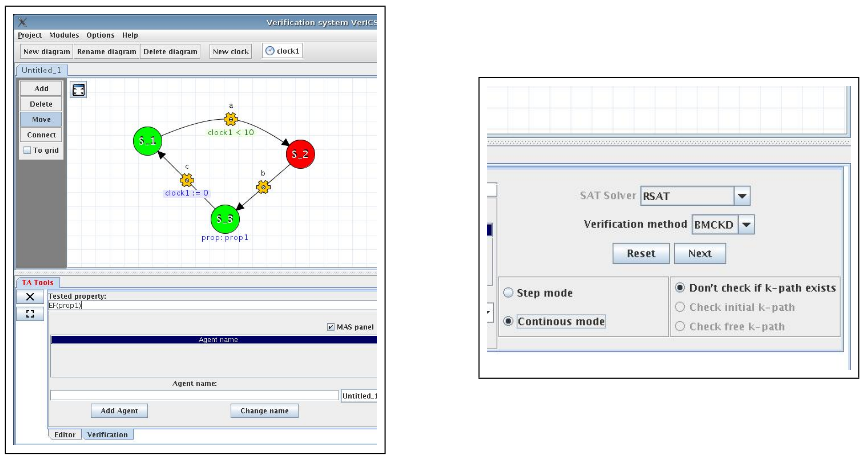
Figure 3. The screenshots of timed automata editor



Cheatnig Dinning Cryptographers Generator
Amount of cryptographers: 3
Cheating cryptographer (0 if none): 1
Tested formula: $\neg Form_1 = \mathrm{EF}(odd \wedge \neg paid_1 \wedge \overline{\mathrm{K}}_1(\bigwedge_{i=2,\ldots,n} \neg paid_i))$
Generate CDC

Bit Transmission Protocol Generator
t1: 2
t2: 3
Tested formula: EF_<0;7>(recack & bit0 & NK[2] (bit0))
Generate BTP

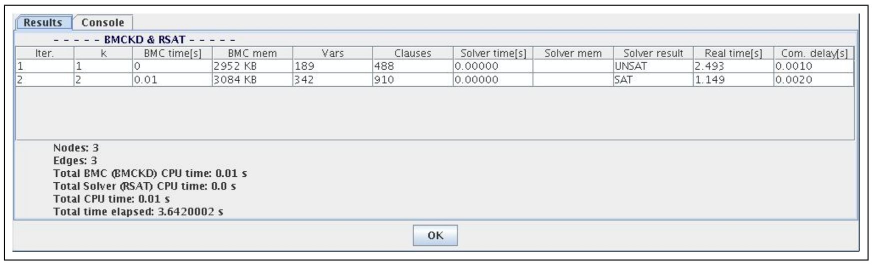
Figure 4. Choosing a scalable benchmark (here Dining Cryptographers) to generate automata automatically



Tested property: EF(prop1)
Agent name: Untitled_1

SAT Solver: RSAT
Verification method: BMCKD
[✓ Don't check if k-path exists] Check initial k-path Check free k-path
Step mode [✓ Continous mode]

Figure 5. The screenshots of the stages of verification (left: editing a formula, right: choosing the verification method and setting its parameters)



- - - - - BMCKD & RSAT - - - - -

Iter.	k	BMC time[s]	BMC mem	Vars	Clauses	Solver time[s]	Solver mem	Solver result	Real time[s]	Com. delay[s]
1	1	0	2952 KB	189	488	0.00000		UNSAT	2.493	0.0010
2	2	0.01	3084 KB	342	910	0.00000		SAT	1.149	0.0020

```
Nodes: 3
Edges: 3
Total BMC (BMCKD) CPU time: 0.01 s
Total Solver (RSAT) CPU time: 0.0 s
Total CPU time: 0.01 s
Total time elapsed: 3.6420002 s
```

Figure 6. The way of presenting the results of verification for BMC and the continuous mode

a single diagram. It is also possible to choose one of the benchmarks available (see Fig 4) to have the automata generated automatically. After introducing the automata, we can test properties of the system. The screenshots on Fig. 5 show some steps of the above process, i.e., editing a property, choosing a verification method and setting up its parameters. Fig. 6 presents the way the results of verification (via BMC) can be provided.

5. Experimental Results

One of the important elements taken into account while rating a model checker is the efficiency of its behaviour. In this section we present some examples of multi-agent and real-time systems we tested, and provide the experimental results obtained. All the examples are standard (scalable) benchmarks, used typically to test efficiency of low-level verification modules. The results we obtain for them allow to compare the effectiveness of VERICS with the other tools available.

The examples we tested were:

- **Dining Cryptographers** - a system consisting of n agents - cryptographers having dinner in a restaurant, who want to know who paid for their meal: one of them or the agency. To reach that while keeping the payer anonymous, each agent flips the coin and states aloud whether the outcomes obtained by him and by the neighbour on his right are equal or different, saying the opposite to what he sees if he has paid. An odd number of differences uttered indicates that a cryptographer was the payer; an even number means that the one who has paid was the agency.

  In order to show the correctness of the above protocol, we prove that the formulas specifying its desired properties hold, while these expressing the undesired ones are false. In our tests we deal with the following formulas:

$$\varphi_D^1 := \mathrm{AG}(odd \wedge \neg paid_1 \Rightarrow \mathrm{K}_1(\bigvee_{i=2,\ldots,n} paid_i)),$$

$$\varphi_D^2 := \mathrm{AG}(odd \wedge \neg paid_1 \Rightarrow \bigvee_{i=2,\ldots,n} \mathrm{K}_1(paid_i)),$$

$$\varphi_D^3 := \mathrm{AG}(\neg paid_1 \Rightarrow \mathrm{K}_1(\bigvee_{i=2,\ldots,n} paid_i)),$$

$$\varphi_D^4 := \mathrm{AG}(even \Rightarrow \hat{K}_n^1(\bigwedge_{i=1,\ldots,n} \neg paid_i)).$$

  The formula φ_D^1 expresses that always when the number of statements "different" is odd and the first cryptographer has not paid for the dinner, then he knows that another cryptographer paid. The formula is proven to hold in the system.

  The next formula φ_D^2, which violates the requirements towards the protocol, says that always when the number of the statements "different" is odd and the first cryptographer has not paid for dinner, then he knows the cryptographer who has paid. The formula is obviously false, since the above information should be secret - none of the cryptographers should know the payer, and an odd number of differences means only that the one who has paid was not the agency.

The third formula specifies the property which means that always when the first cryptographer has not paid for dinner, then he knows that some other cryptographer has paid. This, again, is a false property, as the agency can be the payer as well.

The formula φ_D^4 says that the n-th cryptographer knows that always when the first cryptographer behaves correctly (i.e. follows the protocol) and the number of differences is even, then none of the cryptographers is a payer. Unlike the other properties it is verified in a deontic model in which it is assumed that the first cryptographer may cheat, e.g. he says "equal" while he sees different outcomes of the flips and is not the payer. In this model the formula $\mathrm{AG}(even \Rightarrow \mathrm{K}_n(\bigwedge_{i=1,\dots,n} \neg paid_i))$ is not valid since the even number of differences does not ensure that the agency paid for meal. Therefore, the operator $\hat{K}_n^1$ is used instead of K_n. Now, the formula $\hat{K}_n^1(\bigwedge_{i=1,\dots,n} \neg paid_i)$ expresses that agent n knows that none of the cryptographers paid provided the agent 1 does not cheat. This change makes the whole property true.

The results for the above formulas are presented in Fig. 7. Analysing the results one can notice that the verification techniques applied can be seen as complementary: the UMC method gives better results for the formulas φ_D^1 and φ_D^2, while for φ_D^3 bounded model checking is far more efficient. In the latter case verifying such a big system (up to 1000 cryptographers!) is possible due to the fact that a counterexample can be found on a path which is relatively short. For the first two formulas the length of the path should be incremented up to the model's diameter. In addition to that, the formulas have to be tested on several symbolic paths, which influences the size of the model we are able to test this way.

formula	n	BMC [s/MB]	SAT [s/MB]	n	UMC [s/MB]
φ_D^1	4	0.8 / 9.2	81224 / ⋆	17	367.0 / ⋆
φ_D^2	4	15.97 / 25.8	9359.24 / ⋆	9	392.0 / ⋆
φ_D^3	1000	329 / 1885.0	29.67 / ⋆	15	421.0 / ⋆
φ_D^4	3	0.72 / 7.6	601.14 / ⋆	-	– / ⋆

Figure 7. Experimental results for Dining Cryptographers (the star denotes that no data are available)

- **Fischer's Mutual Exclusion Protocol** - a system consisting of n processes trying to enter their critical sections, and a process which coordinates their access. The behaviour of the system depends on the values δ and Δ ($\delta < \Delta$ makes it incorrect - the mutual exclusion does not hold). The property we have tested is expressed by the formula

$$\varphi_M := \mathrm{EF}\Big(\bigvee_{i,j=1,\dots,n;i\neq j} (crit_i \wedge crit_j)\Big),$$

which states that two different processes can be in their critical sections at the same time (i.e., the mutual exclusion is violated).

The tables in Figures 8 and 9 show the results for verifying the above protocol using the bounded model checking method. In Fig. 8, a counterexample proving that the mutual exclusion does not hold, is found on the path of the length 17. The table in Fig. 9 presents the process of testing unreachability of the property in a system which satisfies the requirements. To this aim, it is checked alternately whether in the system there exists a *free* path [24] of a given length, and whether the property holds on a path of such a length which starts at the initial state of the system. Then, if necessary, the length of the path is incremented. In the case considered in the table it occurred that no free path of length 53 can be found, whereas on all the shorter paths the property tested does not hold. This implies that the property does not hold for the system at all.

	BMC		MiniSat		
k	s	MB	s	MB	satisfiable
0	0.2	4.6	0.0	2.9	no
2	1.0	8.5	0.8	9.8	no
5	2.0	13.8	27.7	37.3	no
8	3.2	19.2	1270.8	521.1	no
11	4.3	24.6	2408.3	852.1	no
14	5.6	29.9	4267.4	1527.7	no
17	7.0	35.3	590.8	419.4	**YES**
Total	23.3	35.3	8565.8	1527.7	

Figure 8. Experimental results for the Fischer protocol with the parameters which make the mutual exclusion violated ($\Delta = 2$, $\delta = 1$). The number of processes involved is 80

	BMC			MiniSat		
k	free path	s	MB	s	MB	satisfiable
0	-	0.0	2.8	0.0	1.7	no
2	+	0.0	3.2	0.0	2.0	yes
2	-	0.0	3.2	0.0	2.1	no
...						
20	+	0.2	6.8	1.6	5.6	yes
20	-	0.2	6.8	5.9	8.8	no
...						
50	+	0.6	12.6	2670.9	206.1	yes
50	-	0.5	12.6	148.1	71.6	no
53	+	0.6	13.2	3992.1	233.1	**NO**
Total		10.4	13.2	10037.9	233.1	

Figure 9. Experimental results for the Fischer protocol with the parameters for which the mutual exclusion holds ($\Delta = 1$, $\delta = 2$). The number of processes involved is 10

Fig. 10 compares the results obtained using the UMC method provided by Verics with these obtained using some other model checkers available. The table shows that in most the cases Verics occurred to be more efficient.

- **Timed Alternating Bit Protocol** - a system consisting of two agents: a sender and a receiver,

Parameters	Time [s]		
	UppAal	RED	VerICS UMC
$N = 10, \Delta = 1, \delta = 2$	37	53	34
$N = 11, \Delta = 1, \delta = 2$	121	141	46
$N = 12, \Delta = 1, \delta = 2$	580	304	59
$N = 13, \Delta = 1, \delta = 2$	-	657	88
$N = 15, \Delta = 1, \delta = 2$	-	-	154
$N = 18, \Delta = 1, \delta = 2$	-	-	376
$N = 20, \Delta = 1, \delta = 2$	-	-	491
$N = 10, \Delta = 3, \delta = 4$	33	53	50
$N = 11, \Delta = 3, \delta = 4$	125	133	71
$N = 10, \Delta = 2, \delta = 1$	7	49	97

Figure 10. A comparison of the results for the Fischer protocol, obtained using VerICS' UMC and the tools UppAal and RED

which exchange messages over two unreliable communication channels, choosing the channel to be used on the basis of the round trip time of a control bit. The properties we have tested were "for each execution of the protocol, if the receiver got an acknowledgement in the time not exceeding t_1 and the value of the bit sent was 0 then the receiver knows the value of this bit" and "for each execution of the protocol if the receiver got an acknowledgement in the time not exceeding t_1 and the value of the bit sent was 0 then the sender knows that the receiver knows the value of this bit", which are expressed, respectively, by the following two formulas:

$$\varphi_A^1 := \mathrm{AG}_{[0,t_1]}((recack \wedge bit0) \Rightarrow \mathrm{K}_R(bit0)),$$

$$\varphi_A^2 := \mathrm{AG}_{[0,t_1]}((recack \wedge bit0) \Rightarrow \mathrm{K}_S\mathrm{K}_R(bit0)).$$

The results of searching for counterexamples for these properties, obtained using the BMC module of VerICS aimed at TECTLK verification, are presented in Fig. 11. In contrast to other benchmarks tested we provide no comparison with other tools, as, to our knowledge, there are no other model checkers which enable TECTLK verification.

6. Final Remarks

In the paper, we presented an overview of the model checker VerICS, focusing on its new features (comparing with the version presented in [3]), i.e., SAT-based verification of multi-agent systems and some extensions and improvements to real-time systems' verification. VerICS offers several capabilities which, to our knowledge, are available in no other tools. It is the only model checker which verifies MAS directly applying SAT-based methods, and the only one handling formulas which combine knowledge and real time. In addition, VerICS is also able to verify RTS. The results for BMC, as well as preliminary results for UMC (obtained using our modification of the SAT-solver ZChaff), seem to be quite promising, taking into account that the systems tested are not optimised w.r.t. the properties to be verified.

	$\neg\varphi_A^1$					$\neg\varphi_A^2$				
	BMC		MiniSat			BMC		MiniSat		
k	s	MB	s	MB	satisfiable	s	MB	s	MB	satisfiable
0	0.0	1.5	0.0	3.8	no	0.0	1.5	0.0	3.8	no
1	0.0	2.3	0.0	4.4	no	0.2	2.8	0.0	4.8	no
2	0.1	3.4	0.0	5.5	no	0.8	4.7	0.1	6.8	no
3	0.2	4.7	0.1	6.8	no	2.3	6.9	0.2	8.5	no
4	0.4	6.4	0.2	8.3	no	6.5	9.6	0.2	11.0	no
5	0.7	9.1	0.1	10.6	no	15.3	14.0	0.3	14.9	no
6	1.0	11.4	0.7	11.9	no	31.8	18.0	1.7	18.2	no
7	1.6	14.2	1.0	14.9	no	57.9	22.6	1.5	21.8	no
8	2.5	17.3	1.6	17.8	no	101.2	27.9	10.5	28.9	no
9	3.3	20.8	1.0	20.6	no	160.5	33.7	10.2	31.1	no
10	4.6	26.6	3.2	25.4	no	274.0	43.4	65.1	50.1	no
11	6.2	31.1	31.3	35.8	**YES**	404.3	51.0	237.5	83.7	**YES**
Total	20.7	31.1	39.2	35.8		1054.6	51.0	327.3	83.7	

Figure 11. Experimental results for the timed alternating bit protocol

References

[1] R. Bordini, M. Fisher, C. Pardavila, W. Visser, and M. Wooldridge. Model checking multi-agent programs with CASP. In *Proc. of the 15th Int. Conf. on Computer Aided Verification (CAV'03)*, volume 2725 of *LNCS*, pages 110–113. Springer-Verlag, 2003.

[2] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[3] P. Dembiński, A. Janowska, P. Janowski, W. Penczek, A. Półrola, M. Szreter, B. Woźna, and A. Zbrzezny. VerICS: A tool for verifying timed automata and Estelle specifications. In *Proc. of the 9th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *LNCS*, pages 278–283. Springer-Verlag, 2003.

[4] E. A. Emerson and E. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *Proc. of the 7th Int. Colloquium on Automata, Languages and Programming (ICALP'80)*, volume 85 of *LNCS*, pages 169–181. Springer-Verlag, 1980.

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

[6] P. Gammie and R. van der Meyden. MCK: Model checking the logic of knowledge. In *Proc. of the 16th Int. Conf. on Computer Aided Verification (CAV'04)*, volume 3114 of *LNCS*, pages 479–483. Springer-Verlag, 2004.

[7] ISO/IEC 9074(E), Estelle - a formal description technique based on an extended state-transition model. International Standards Organization, 1997.

[8] M. Kacprzak, A. Lomuscio, T. Łasica, W. Penczek, and M. Szreter. Verifying multiagent systems via unbounded model checking. In *Proc. of the 3rd NASA Workshop on Formal Approaches to Agent-Based Systems (FAABS III)*, volume 3228 of *LNCS*, pages 189–212. Springer-Verlag, 2005.

[9] M. Kacprzak, A. Lomuscio, A. Niewiadomski, W. Penczek, F. Raimondi, and M. Szreter. Comparing BDD and SAT based techniques for model checking Chaum's dining cryptographers protocol. *Fundamenta Informaticae*, 72(1-2):215–234, 2006.

[10] M. Kacprzak, A. Lomuscio, and W. Penczek. From bounded to unbounded model checking for temporal epistemic logic. *Fundamenta Informaticae*, 63(2-3):221–240, 2004.

[11] A. Lomuscio, W. Penczek, and B. Woźna. Bounded model checking for knowledge and real time. *Artificial Intelligence*, 171:1011–1038, 2007.

[12] A. Lomuscio and F. Raimondi. MCMAS: A model checker for multi-agent systems. In *Proc. of the 12th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, volume 3920 of *LNCS*, pages 450–454. Springer-Verlag, 2006.

[13] A. Lomuscio, B. Woźna, and A. Zbrzezny. Bounded model checking real-time multi-agent systems with clock differences: Theory and implementation. In *Proc. of the 4th Int. Workshop on Model Checking and Artificial Intelligence (MoChArt'06)*, pages 62–78. ECCAI, 2006.

[14] K. L. McMillan. Applying SAT methods in unbounded symbolic model checking. In *Proc. of the 14th Int. Conf. on Computer Aided Verification (CAV'02)*, volume 2404 of *LNCS*, pages 250–264. Springer-Verlag, 2002.

[15] MiniSat. http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat, 2006.

[16] W. Penczek and A. Lomuscio. Verifying epistemic properties of multi-agent systems via bounded model checking. In *Proc. of the 2nd Int. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS'03)*, pages 209–216. ACM, 2003.

[17] W. Penczek and A. Półrola. *Advances in Verification of Time Petri Nets and Timed Automata: A Temporal Logic Approach*, volume 20 of *Studies in Computational Intelligence*. Springer-Verlag, 2006.

[18] RSat. http://reasoning.cs.ucla.edu/rsat, 2006.

[19] M. Szreter. *SAT-Based Model Checking of Distributed Systems*. PhD thesis, ICS PAS, January 2007.

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

[21] M. Wooldridge, M. Fisher, M. P. Huget, and S. Parsons. Model checking multiagent systems with MABLE. In *Proc. of the 1st Int. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS'02)*, volume II, pages 952–959. ACM, 2002.

[22] B. Woźna, A. Lomuscio, and W. Penczek. Bounded model checking for deontic interpreted systems. In *Proc. of the 2nd Int. Workshop on Logic and Communication in Multi-Agent Systems (LCMAS'04)*, volume 126 of *ENTCS*, pages 93–114. Elsevier, 2005.

[23] B. Woźna, A. Lomuscio, and W. Penczek. Bounded model checking for knowledge and real time. In *Proc. of the 4th Int. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS'05)*, pages 165–172. ACM, 2005.

[24] A. Zbrzezny. Improvements in SAT-based reachability analysis for timed automata. *Fundamenta Informaticae*, 60(1-4):417–434, 2004.

[25] A. Zbrzezny. SAT-based reachability checking for timed automata with diagonal constraints. *Fundamenta Informaticae*, 67(1-3):303–322, 2005.

[26] L. Zhang. Zchaff. http://www.ee.princeton.edu/∼chaff/zchaff.php, 2001.