Automatic Verification of Knowledge and Time with NuSMV

Alessio Lomuscio*
Dept. of Computing
Imperial College London, UK
A.Lomuscio@imperial.ac.uk

Charles Pecheur†
Dept. INGI
Univ. cath. de Louvain, Belgium
charles.pecheur@uclouvain.be

Franco Raimondi‡
Dept. of Computer Science
Univ. College London, UK
f.raimondi@cs.ucl.ac.uk

Abstract

We show that the problem of model checking multi-dimensional modal logics can be reduced to the problem of model checking ARCTL, an extension of the temporal logic CTL with action labels and operators to reason about actions. In particular, we introduce a methodology for model checking a temporal-epistemic logic by building upon an extension of the model checker NuSMV that enables the verification of ARCTL. We briefly present the implementation and report experimental results for the verification of a typical security protocol involving temporal-epistemic properties: the protocol of the dining cryptographers.

1 Introduction

Epistemic logic [Fagin *et al.*, 1995; Meyer and Hoek, 1995] has traditionally played an important role in Artificial Intelligence (AI). Not only does epistemic logic provide a formal basis for reasoning about states of knowledge in automatic reasoners but it can also be seen as a formal specification language to reason about artificial agents. Indeed, epistemic concepts are particularly appropriate in multi-agent systems (MAS) as knowledge constitutes the basis for rational action. In this line of work typically one considers the epistemic modal logic $S5_n$ combined with a temporal logic for branching time or linear time interpreted on computationally grounded semantics [Wooldridge, 2000] such as the one of interpreted systems [Fagin *et al.*, 1995], or a suitable variation of it [Wooldridge and Lomuscio, 2000].

While much attention in the 80s and 90s focused on proving meta-logical results (notably completeness and computational complexity) for various temporal and epistemic combinations [Halpern and Moses, 1992; Meyden, 1994; Meyden and Wong, 2003], considerable attention has been given in the past few years to the problem of devising model checking techniques for these formalisms [Hoek and Wooldridge, 2002; Gammie and van der Meyden, 2004; Raimondi and Lomuscio, 2005; Penczek and Lomuscio, 2003]. These efforts are strictly related to the recent shift of attention from theorem proving to model checking as suggested, among others, in [Halpern and Vardi, 1991].

This article intends to make a contribution in this line by proposing an efficient model checking technique for verifying CTLK (an epistemic logic on branching time [Penczek and Lomuscio, 2003]) based on NuSMV [Cimatti *et al.*, 2002], a mainstream model checker for temporal logic. Specifically the present article makes the following points. First, it is shown that the model checking of CTLK can be (automatically) rephrased as the one of model checking the action-based temporal logic ARCTL [Pecheur and Raimondi, 2006]. Second, we present an automatic translator from an SMV-like language for a semantics of MAS for the extension above. The main benefits of this proposal against the current state-of-the-art lies in the efficiency of the approach which we try to demonstrate by discussing experimental results.

2 Preliminaries

We summarise the formalism of *interpreted systems* in Section 2.1, a formalism to reason about time and knowledge in a system of agents. In Section 2.2 we discuss the problem of *model checking* using NuSMV.

2.1 Interpreted systems and CTLK

The formalism of interpreted systems, introduced in [Fagin *et al.*, 1995], provides a formal framework to reason about time and knowledge in a system of agents. Let Σ be a set of n agents: $\Sigma = \{1, \dots, n\}$. A set of local states L_i and a set of actions Act_i is associated to each agent i, together with a protocol $P_i : L_i \to 2^{Act_i}$ assigning a list of enabled actions to each local state. The local states of an agent change according to a local evolution function $t_i : L_i \times L_E \times Act \to L_i$, where L_E is the set of local states of a special agent in Σ (the environment), and $Act = Act_1 \times \dots \times Act_n$. The set $S = L_1 \times \dots \times L_n$ is called the set of global states. Given a set of initial global states $I \subseteq S$, the set $G \subseteq S$ representing the set of reachable states is generated by the evolution of I and in accordance with the protocol and the agents' local evolution functions[1]. Given a set of atomic propositions AP

*The author acknowledges support from EPSRC grant GR/S49353.

†With RIACS at NASA Ames while performing this work.

‡With MCT at NASA Ames while performing this work. The author acknowledges support from EPSRC grant CNA/04/04.

[1] The system evolves synchronously, i.e, at each time step all the agents perform a move.

and an interpretation $V \subseteq S \times AP$, an *interpreted system* is a tuple

$$IS = \left\langle (L_i, Act_i, P_i, t_i)_{i \in \Sigma}, I, V \right\rangle$$

The logic CTLK combines the traditional AI epistemic logic S5$_n$ with the temporal logic CTL. Specifically, the syntax of CTLK is defined as follows:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid EX\varphi \mid EG\varphi \mid E[\varphi U \varphi] \mid K_i\varphi \quad (1)$$

where $K_i\varphi$ is read as "agent i knows φ" and the CTL operators have the standard meaning — for example, $EG\varphi$ is read as "there exists a path where φ holds forever". Other derived operators are defined in a standard way (see e.g. [Clarke *et al.*, 1999; Fagin *et al.*, 1995]). To evaluate CTLK formulae, a Kripke model $M_{IS} = (W, I, R_t, \sim_1, \ldots, \sim_n, V)$ is associated to a given interpreted system IS, as follows: W is the set G of reachable states, $I \subseteq W$ is the set of initial states, the temporal relation $R_t \subseteq W \times W$ is obtained using the protocols P_i and the evolutions functions t_i, the epistemic relations $\sim_i \subseteq W \times W$, for $i \in \Sigma$, are defined by checking the equality of the i-th local component of two global states (i.e., $(l_1, \ldots, l_n) \sim_i (l'_1, \ldots, l'_n)$ iff $l_i = l'_i$), and V is the evaluation relation appearing in IS[2].

The formulae defined by (1) are interpreted in M_{IS} in a standard way: we refer to [Clarke *et al.*, 1999; Fagin *et al.*, 1995; Penczek and Lomuscio, 2003] for the formal definition of $M \models \varphi$, where $\models$ is the standard satisfaction relation.

2.2 Model checking using NuSMV

Given a Kripke model M and a formula φ, *model checking* is defined as the problem of establishing whether or not $M \models \varphi$. In this approach M represents the system to be checked and φ the specification of the system. In the last fifteen years techniques and tools have been developed to perform the verification task in an automatic way, mainly for temporal models and temporal specifications. NuSMV [Cimatti *et al.*, 2002] is a mature model checker for temporal logics and it has been employed in the verification of a number of examples. Other very successful model checkers exist, notably Spin [Holzmann, 1997], Verics [Nabialek *et al.*, 2004], etc.

NuSMV has a dedicated modelling language (the SMV language), which permits the definition of the temporal model in an expressive, compact and modular way. NuSMV avoids building or exploring the state space corresponding to its models explicitly; instead, NuSMV applies symbolic techniques based on ordered binary decision diagrams (OBDDs) or propositional satisfiability (SAT) solvers to efficiently perform verification over large state spaces.

NuSMV is a command line tool available for most operating systems, and its source code is available under the terms of the GNU General Public License (GPL).

3 Model checking MAS: state of the art

Recently, different approaches have been proposed to extend model checking techniques from temporal logics to richer logics, with the aim of verifying knowledge-based and agent-based systems. Two main streams can be identified in the recent literature:

1. **Dedicated tools**. Works along this line include the model checker MCK (Model Checking Knowledge, [Gammie and van der Meyden, 2004]), implementing the verification of certain classes of interpreted systems. Verics [Nabialek *et al.*, 2004] is a model checker for MAS described using networks of timed automata. MCMAS [Raimondi and Lomuscio, 2005] is an OBDD-based model checker for MAS described in interpreted systems.
2. **Extensions (and translations) to existing tools**. [Hoek and Wooldridge, 2002] propose the use of local propositions to reduce the problem of model checking knowledge and time to the verification of a temporal-only model. [Wooldridge *et al.*, 2002] define the MABLE language, and they show how the verification of this language can be reduced to the verification of PROMELA code (the input language of the model checker SPIN [Holzmann, 1997]). Similarly, [Bordini *et al.*, 2003] introduce the language AgentSpeak(F), and they present a translation into PROMELA code.

Experimental results from the papers cited above show that, on average, purpose-built tools can handle larger examples, and that trying to use existing tools often requires manual intervention. We show below this is not necessarily the case. In the remainder of this paper our aim is to introduce a fully automated methodology that builds upon an existing tool (NuSMV) and show that it performs comparably or better than similar approaches.

4 Model checking CTLK in NuSMV

This section introduces the logic ARCTL (Action-Restricted CTL) and a proposed extension to NuSMV thereby enabling the verification of ARCTL operators. In Section 4.2 we show how the problem of model checking for CTLK can be reduced to the problem of model checking for ARCTL, thereby permitting the verification of CTLK by using NuSMV. In addition, in Section 4.3 we present an SMV-like language for interpreted systems and its translation into SMV code.

4.1 The logic ARCTL

The logic ARCTL [Pecheur and Raimondi, 2006] extends the logic CTL by allowing quantification over action labelled paths. More in detail, given a set of atomic propositions AP and a set of atomic actions AA, the syntax of ARCTL is defined as:

$$\begin{aligned} \varphi \quad &::= \quad p \mid \neg\varphi \mid \varphi \vee \varphi \mid E_\alpha X\varphi \mid A_\alpha X\varphi \mid \\ & \qquad E_\alpha[\varphi U \varphi] \mid A_\alpha[\varphi U \varphi] \qquad (2) \\ \alpha \quad &::= \quad b \mid \neg\alpha \mid \alpha \vee \alpha \qquad (3) \end{aligned}$$

where φ and α are state and action formulae and $p \in AP, b \in AA$ are atomic propositions over states and actions, respectively. Similarly to CTL, other temporal operators can be derived in a standard way.

[2] Note that W is, by definition, the *reachable* state space, i.e. it only contains states that are reachable from I through R_t. This condition is crucial for the proper interpretation of $\sim_i$ and $K_i\varphi$.

A model for ARCTL is a tuple of the form $M = (S, S_0, A, T, V_P, V_A)$, where S is a set of states, $S_0 \subseteq S$ is a set of initial states, A is a set of actions, $T \subseteq S \times A \times S$ is a transition relation (notice the dependence on actions), $V_P : S \rightarrow 2^{AP}$ is an interpretation for atomic propositions, and $V_A : A \rightarrow 2^{AA}$ is an interpretation for atomic actions. Given a model $M = (S, S_0, T, V_P, V_A)$, the α-restriction of M, denoted by M_α, is a model $M_\alpha = (S, S_0, T_\alpha, V_P, V_A)$, where T_α is a transition relation such that $(s, a, s') \in T_\alpha$ iff $(s, a, s') \in T$ and $a \models \alpha$ (where $\models$ is the natural extension of V_A to propositional formulae α). We refer to [Pecheur and Raimondi, 2006] for further details.

For model checking purposes, we have extended NuSMV to support the verification of ARCTL formulae. We used NuSMV existing "input" variables to model ARCTL actions. In particular, we have modified the syntax of the formulae accepted by NuSMV as follows:

$$\begin{array}{lll} \mathit{ctlexpr} & ::= & \ldots \quad \text{(existing CTL forms)} \\ & & \texttt{EAX(}\mathit{simpleexpr}\texttt{)}\,\mathit{ctlexpr} \\ & & \texttt{EAG(}\mathit{simpleexpr}\texttt{)}\,\mathit{ctlexpr} \\ & & \texttt{EA(}\mathit{simpleexpr}\texttt{)}\,\texttt{[}\mathit{ctlexpr}\ \texttt{U}\ \mathit{ctlexpr}\texttt{]} \end{array}$$

where $\mathit{simpleexpr}$ is a conditional expression, further restricted to contain only input variables. For example, `EA(a)[p U q]` is the concrete syntax for $E_a[pUq]$. Additionally, we have implemented extensions to the NuSMV code base to enable the verification of these operators (the details of these modifications are beyond the scope of this paper, we refer to [Pecheur and Raimondi, 2006] for further details).

4.2 Reducing CTLK to ARCTL

The problem of model checking CTLK (see Section 2.1) can be reduced to the problem of model checking ARCTL. Specifically, given a CTLK model M_K and a CTLK formula φ_K, we can define an ARCTL model $M = F(M_K)$ and an ARCTL formula $F(\varphi_K)$ such that $M_K \models \varphi_K$ iff $F(M_K) \models F(\varphi_K)$. Let $\Sigma = \{1, \ldots, n\}$ be a set of agents, and let $M_K = (W, I, R_t, \{\sim_i\}_{i \in \Sigma}, V)$ be a model associated to some interpreted system $IS = \langle (L_i, Act_i, P_i, t_i)_{i \in \Sigma}, I, V \rangle$. The model $F(M_K)$ is an ARCTL model $M = (S, S_0, A, T, V_P, V_A)$ such that

- $S = W$ and $S_0 = I$;
- the set $AA = \{Run, Agt_1, \ldots, Agt_n\}$ contains a proposition Run to label temporal transitions (defined by R_t) and n propositions Agt_i (one for each agent) to label epistemic equivalence steps (defined by $\sim_i$), and the action set A is 2^{AA};
- the transition relation T combines the temporal transition R_t and the epistemic relations $\{\sim_i\}_{i \in \Sigma}$ in the following way: for states $s, s' \in W$, (i) $(s, \{Run\}, s') \in T$ iff sR_ts'; (ii) $(s, \{Agt_i\}, s') \in T$ iff $s \sim_i s'$; (iii) $(s, \{a1, \ldots, a_k\}, s') \in T$ iff $(s, \{a_i\}, s')$ for all $1 \leq i \leq k$. [3]

[3] Case (iii) is necessary for interpreting distributed knowledge operators, but this discussion is beyond the scope of this paper.

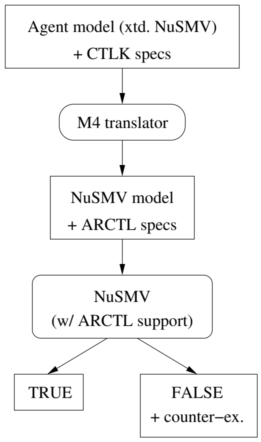

Figure 1: Verification work flow for interpreted systems

The translation of a temporal-epistemic formula into an ARCTL formula is inductively defined as follows:

- $F(p) = p$, if p is a propositional formula.
- $F(EX\varphi) = E_{Run}X\varphi$; $F(E[\varphi U \psi]) = E_{Run}[\varphi U \psi]$; $F(EG\varphi) = E_{Run}G\varphi$; $F(K_i\varphi) = A_{Agt_i}X\varphi$.

In other words we use the labels Run (Agt_i, respectively) to denote a temporal relation (the epistemic relation for agent i, respectively). This translation allows us to model check CTLK formulas by model checking their translations in ARCTL. Clearly a similar approach can be used for more complex modal logics.

The translation from an interpreted system IS to the ARCTL model $F(M_{IS})$ is performed automatically by a translator we have implemented (see next section).

4.3 An SMV-like language for interpreted systems

We have designed extensions of the SMV language for the description of interpreted systems and CTLK formulae. These extensions can be translated to into a standard SMV model and ARCTL formulae. Concretely, the extensions are defined as a library of M4 macros[4]. The work flow needed to perform verification of an interpreted system is summarised in Figure 1. Notice that the only manual intervention is the provision of the input file describing the problem to be verified.

In this language, an agent <name> is associated with the SMV variables <v1>,..,<vn> that define its local state, through a declaration AGENT(<name>,<v1>,..,<vn>). The actions of each agent are represented as input variables (IVAR). The protocol of each agent is described as a relation between its local state and action variables (within an SMV `TRANS` statement). The transition function is encoded using a new `TTRANS` statement, and initial conditions using a new `TINIT` statement. Figure 2 shows the structure of a typical definition of a class of agents as an SMV module (note that "`--`" starts comments in SMV, see [Cimatti *et al.*, 2002] for details).

[4] M4 [Kernighan and Ritchie, 1977] is a general-purpose macro processor available on most UNIX platforms.

```
MODULE anAgent(args,ENV) -- an agent module
  VAR local : {...};     -- the local state
  AGENT(me,local)        -- declare the local state
  IVAR action : {...};   -- actions of the agent
  TINIT( ... );          -- initial conditions
  TRANS( action =
         case ... )      -- the agent's protocol
  TTRANS( NEXT(local) =
          case ... )     -- the agent's evolution function

MODULE main              -- main module
VAR_ENV                  -- declare ENV variables
VAR alice : anAgent(args1,ENV) ;  -- an agent
    bob   : anAgent(args2,ENV) ;  -- another agent
```

Figure 2: Example of agent definition in extended SMV

Internally, the translation generates additional Boolean variables IVARS corresponding to the Run and Agt_i propositions of the ARCTL model (e.g. `RUN`, `bob.me` and `alice.me` in the model of Fig. 2), and `TTRANS` statements expand to standard `TRANS` statements conditioned on `RUN`.

Note that the state space of the NuSMV model is not a priori restricted to the (temporally) reachable states; this has to be imposed by the translation scheme. A state is reachable iff it can be reached from the initial states through a series of temporal steps (or, equivalently, iff there exists a reverse path from that state back to the initial set). Assuming we have access to reverse temporal transitions through some action condition $Back$ in our ARCTL model, and a state condition $Init$ for initial states, the set of the (temporally) reachable states is captured by the following ARCTL formula:

$$Reachable \equiv E_{Back} F \; Init$$

Since this is a temporal formula, it has to be folded into NuSMV properties; it cannot be used in the NuSMV model itself. For example, the CTLK formula $K_i\varphi$ expands to the ARCTL formula $A_{Agt_i}X(Reachable \rightarrow \varphi)$.

Our library implements this scheme. In particular, the `TTRANS` construct implements an encoding scheme allowing temporal transitions to be traversed both forwards and backwards. On the specification side, new operators `TAX`, `TAG`, etc. provide the equivalent of SMV's built-in `AX`, `AG`, etc., restricted to temporal transitions, and a new operator `KK` implements $K_i\varphi$. For instance, the CTLK formula $AG(K_a(p \vee q))$ expressing that agent a always knows either p or q, is written as `TAG(KK(a,p|q))`.

5 Example application

In this section we model the protocol of the dining cryptographers using the formalism presented in Section 2.1, in order to enable its verification using the methodology presented in Section 4.2. The protocol was introduced by Chaum in [Chaum, 1988], and was modelled using agents by various authors [van der Meyden and Su, 2004; Raimondi and Lomuscio, 2005; Kacprzak *et al.*, 2006]. The aim of this protocol is to allow the anonymous broadcasting of messages, and it is usually introduced using the following scenario (wording from [Chaum, 1988]):

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

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

This protocol can be extended to any number n of cryptographers greater than three. We encode the scenario by using an interpreted system by introducing n agents $C_1, \ldots, C_n$, and one agent E to represent the environment (which selects the payer and the result of coin tosses at the beginning of each run, non-deterministically) [Raimondi and Lomuscio, 2005].

The local state of cryptographer C_i is modelled using three variables $\mathtt{equal_i}, \mathtt{paid_i}, \mathtt{even_i}$, representing respectively whether the coins that C_i can see are equal or different, whether C_i is the payer, and whether the number of different utterances reported is even.[5] The list of actions for each cryptographer includes the actions `do nothing`, `say equal`, and `say different`, performed in compliance with the description provided above. In the initial state the variables for each cryptographer are initialised to a null value, and they are updated in the first time step, to reflect the environment's configuration. At this point, each cryptographer utters the appropriate phrase (represented by `say equal` or `say different`) and the variable storing the value of "different" utterances (either even or odd) is appropriately updated.

The key properties of this scenario [van der Meyden and Su, 2004] are easy to express using CTLK. For instance:

$$\begin{array}{c}(\mathtt{odd} \wedge \neg \mathtt{paid}_1) \rightarrow AX(K_{C1}(\mathtt{paid}_2 \vee \mathtt{paid}_3) \\ \wedge \neg K_{C1}(\mathtt{paid}_2) \wedge \neg K_{C1}(\mathtt{paid}_3))\end{array} \quad (4)$$

$$\mathtt{even} \rightarrow AX(K_{C1}(\neg \mathtt{paid}_2 \wedge \neg \mathtt{paid}_3)) \quad (5)$$

Formula (4) expresses the property that, if the first cryptographer did not pay for the dinner and there is an odd number of utterances, then, upon update of his local state, the first cryptographer *knows* that someone of the remaining cryptographers paid for the dinner, but the first cryptographer does not know who the payer is. Formula (5) expresses the property that if the number of utterances is even, the first cryptographers knows that nobody paid for the dinner.

[5] We refer to [Kacprzak *et al.*, 2006] for other possible encodings of the same protocol. Our choice here is motivated by the need of comparing our experimental results with the ones in [Raimondi and Lomuscio, 2005; Kacprzak *et al.*, 2006].

N. crypt	Bool vars	NuSMV	MCMAS	VERICS
3	61	0.34s	0.67s	84s
4	78	0.46s	2.09s	730s
5	99	0.86s	8.91s	1h5m
6	116	2.95s	19.5s	8h32m
7	137	1m15s	2m29s	N/A
8	156	6m34s	2h59m	N/A
9	175	49m5s	N/A	N/A

Table 1: Average experimental results.

We have encoded this scenario using the language presented in Section 4.3 and we have been able to verify the example for up to nine cryptographers[6]. Experimental results are reported in the next section.

6 Experimental results

Table 1 reports the time results obtained in the verification of the example presented in the previous section, as a function of the number of cryptographers (first column). The third column reports the time required for the verification of the formulae 4 and 5 (appropriately translated into ARCTL formulae). The second column reports the number of Boolean variables required to encode the example (see [Clarke *et al.*, 1999] for more details on this technique), and provides an estimate for the size of the model; for instance, 137 Boolean variables are required to encode an example with 7 cryptographers, corresponding to a maximal state space of size $2^{137} \approx 10^{41}$.

The fourth and fifth columns report the time results obtained in the verification of the protocol of the dining cryptographers using the model checkers MCMAS and Verics, as reported in [Raimondi and Lomuscio, 2005; Kacprzak *et al.*, 2006], for the verification of the same formulae mentioned above, as a function of the number of cryptographers. Notice that, in the case of MCMAS, the verification time does not depend on the formula being verified and it is usually a fraction of the time spent in reading and parsing the model (the opposite is true for NuSMV, because of different implementation choices). The last column reports the time results for Verics. Differently from the previous two cases, Verics's results are based on an implementation of Bounded Model Checking for CTLK. [Penczek and Lomuscio, 2003]. Due to this, Verics's performance for this two formulae is worse than the other two model checkers, but Verics is normally capable of finding counterexamples for false formulae effectively: in [Kacprzak *et al.*, 2006] it is shown that certain false formulae can be verified in scenarios with up to 100 cryptographers. Neither MCMAS nor the technique of this article can handle this magnitude of state spaces.

6.1 Discussion

A comparison of the results obtained with the three model checkers is reported in Figure 3. Clearly for the formulae considered here the approach presented here performs moderately better than MCMAS and VERICS. The better performance is due to the optimisation techniques implemented in NuSMV (such as on-the-fly model checking and caching). Obviously, experimental results depend on the examples tested and it is not appropriate to draw final conclusions on one example only. Additionally, we would expect VERICS to outperform the approach presented here when trying to falsify formulas on very large state spaces. Nevertheless, we think the discussion above shows that the technique presented here can significantly complement the other model checkers in many instances.

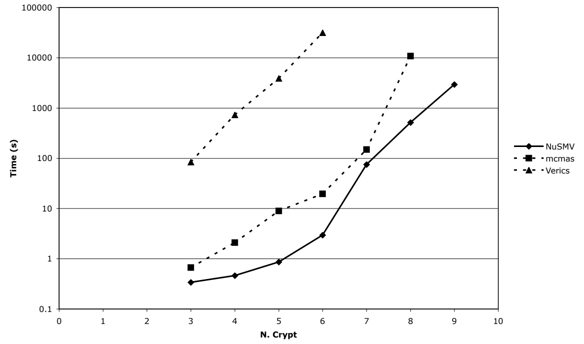

Figure 3: Comparison of the experimental results.

We could not compare our methodology to other existing approaches based on existing model checkers (see Section 3). Indeed, such approaches often require a manual intervention in the translation from their specific programming language into the programming language of the temporal model checkers, and this is not feasible for large examples such as the one considered here.

7 Conclusion

In this paper we have presented a novel technique for model checking CTLK that relies on the translation of this logic into action-based temporal logic and model checking of this logic with NuSMV. The approach is sound and complete and our implementation shows experimental results that are in line with or better than existing specialised tools.

The use of macros for extending the NuSMV language has allowed for easy prototyping but limits the syntactic flexibility. A more natural syntax could be supported with more involved translation facilities. Further planned work includes investigating optimisation of the verification scheme, both at the level of the translation from CTLK to ARCTL and through additional extensions or optimizations of NuSMV itself. Moreover, we limited our analysis to dining cryptographers because this is the only scenario for which experimental results are available in the literature. In future work we certainly plan to use the technique and tool presented here to investigate more realistic scenarios.

Another important issue to be addressed is the handling of witness traces generated by NuSMV, which need to be formulated back in terms of the original CTLK model. We would also like to investigate using NuSMV's SAT-based bounded model checking capabilities rather than the current OBDD-based approach. Given the limitation of NuSMV, this requires

[6]The source code for these examples is available from `http://www.cs.ucl.ac.uk/staff/f.raimondi/dincrypt-code.zip`.

shifting from *branching* to *linear* temporal logic, requiring a new and more restrictive encoding scheme for CTLK properties. The feasibility and applicability of such a scheme remains to be explored.

References

[Bordini *et al.*, 2003] R. H. Bordini, M. Fisher, C. Pardavila, and M. Wooldridge. Model checking AgentSpeak. In J. S. Rosenschein, T. Sandholm, W. Michael, and M. Yokoo, editors, *Proceedings of AAMAS-03*, pages 409–416. ACM Press, 2003.

[Chaum, 1988] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.

[Cimatti *et al.*, 2002] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NUSMV2: An open-source tool for symbolic model checking. In *Proceedings of CAV'02*, volume 2404 of *LNCS*, pages 359–364. Springer-Verlag, 2002.

[Clarke *et al.*, 1999] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

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

[Gammie and van der Meyden, 2004] P. Gammie and R. van der Meyden. MCK: Model checking the logic of knowledge. In *Proceedings of CAV'04*, volume 3114 of *LNCS*, pages 479–483. Springer-Verlag, 2004.

[Halpern and Moses, 1992] J. Halpern and Y. Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54:319–379, 1992.

[Halpern and Vardi, 1991] J. Halpern and M. Vardi. Model checking vs. theorem proving: A manifesto. In *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 325–334. Morgan Kaufmann, April 1991.

[Hoek and Wooldridge, 2002] W. van der Hoek and M. Wooldridge. Model checking knowledge and time. In *SPIN 2002 – Proceedings of the Ninth International SPIN Workshop on Model Checking of Software*, Grenoble, France, April 2002.

[Holzmann, 1997] G. J. Holzmann. The model checker SPIN. *IEEE transaction on software engineering*, 23(5):279–295, 1997.

[Kacprzak *et al.*, 2006] 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*, 2006. to appear.

[Kernighan and Ritchie, 1977] B.W. Kernighan and D.M. Ritchie. *The M4 Macro Processor*. Bell Laboratories, 1977.

[Meyden and Wong, 2003] R. van der Meyden and K. Wong. Complete axiomatizations for reasoning about knowledge and branching time. *Studia Logica*, 75(1):93–123, 2003.

[Meyden, 1994] R. van der Meyden. Axioms for knowledge and time in distributed systems with perfect recall. In *Proceedings, Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 448–457, Paris, France, 1994. IEEE Computer Society Press.

[Meyer and Hoek, 1995] J.-J. Ch. Meyer and W. van der Hoek. *Epistemic Logic for AI and Computer Science*, volume 41 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1995.

[Nabialek *et al.*, 2004] W. Nabialek, A. Niewiadomski, W. Penczek, A. Pólrola, and M. Szreter. VERICS 2004: A model checker for real time and multi-agent systems. In *Proceedings of CS&P'04*, volume 170 of *Informatik-Berichte*, pages 88–99. Humboldt University, 2004.

[Pecheur and Raimondi, 2006] C. Pecheur and F. Raimondi. Symbolic model checking of logics with actions. In *Proceedings of MoChArt 2006*, Lecture Notes in Artificial Intelligence. Springer Verlag, August 2006. to appear.

[Penczek and Lomuscio, 2003] W. Penczek and A. Lomuscio. Verifying epistemic properties of multi-agent systems via bounded model checking. *Fundamenta Informaticae*, 55(2):167–185, 2003.

[Raimondi and Lomuscio, 2005] F. Raimondi and A. Lomuscio. Automatic verification of multi-agent systems by model checking via OBDDs. *Journal of Applied Logic*, 2005. To appear in Special issue on Logic-based agent verification.

[van der Meyden and Su, 2004] R. van der Meyden and Kaile Su. Symbolic model checking the knowledge of the dining cryptographers. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop (CSFW'04)*, pages 280–291, Washington, DC, USA, 2004. IEEE Computer Society.

[Wooldridge and Lomuscio, 2000] M. Wooldridge and A. Lomuscio. Multi-agent $\mathcal{VSK}$ logic. In M. Ojeda-Aciego, I. P. de Guzmán, G. Brewka, and L. Moniz Pereira, editors, *Proceedings of JELIA 2000 (LNAI Volume 1919)*, pages 300–312. Springer-Verlag, 2000.

[Wooldridge *et al.*, 2002] M. Wooldridge, M. Fisher, M. Huget, and S. Parsons. Model checking multiagent systems with MABLE. In *Proceedings of the First International Conference on Autonomous Agents and Multiagent Systems (AAMAS-02)*, pages 952–959, Bologna, Italy, July 2002.

[Wooldridge, 2000] M. Wooldridge. Computationally grounded theories of agency. In E. Durfee, editor, *Proceedings of ICMAS, International Conference of Multi-Agent Systems*, pages 13–22. IEEE Press, 2000.