Model Checking a Knowledge Exchange Scenario

Sieuwert van Otterloo Wiebe van der Hoek Michael Wooldridge
University of Liverpool
Department of Computer Science
Liverpool L69 7ZF, UK
tel 0151 794 3706 fax 0151 794 3715
{sieuwert,wiebe,mjw}@csc.liv.ac.uk

November 12, 2004

Abstract

We are interested in applying model checking techniques to the verification of communication protocols which require safe communication. Typically, in such scenarios, one desires to demonstrate that one party can reliably communicate information to another party without a third party being able to determine this information. Our approach involves using the modal logic of knowledge, which has only relatively recently been studied in the context of secure protocols. We demonstrate our approach by means of a detailed case study: The Russian cards problem. This is an example of a security protocol with nontrivial requirements on the knowledge of the agents involved. Using the Russian cards problem as an example it is shown how the satisfaction of properties involving knowledge can be verified in a standard model checker — in our case, SPIN.

1 Introduction

Verification of multi agent systems and protocols is an interesting problem, because multi agent systems often have complicated requirements that involve knowledge, time, strategies and other agent notions. In this paper we are interested in the automatic verification of protocols involving knowledge. We concentrate on a recently discovered method for model checking knowledge and time and apply this method to an example problem, the Russian Cards Problem [13].

The Russian Cards Problem contains two typical requirements: on the one hand it states that successful communication between two agents should occur, while at the same time certain information must remain secret to a third agent. This is fairly typical for security protocols in which secrecy is required. One way to obtain secrecy is to use encryption. The security is then based on the inability of the third agent to break the encryption, and is thus based on the computational limitations, in terms of time and

memory, of the third agents. This can be called computation based security (secrecy is of course only one aspect of security, but in this paper we are mainly concerned with secrecy and ignore other aspects such as identification, authentification and availability). We are not concerned with encryption in this paper, but rather focus on the knowledge that agents could obtain when they have unbounded computing power. The secrecy is thus not computation based but information based.

To specify epistemic properties — properties pertaining to the knowledge of communication participants — one can make use of a by now well-accepted modal logic for knowledge [9, 4], which provides modal operators K_i to represent what a specific agent i knows, and group modalities E and C to represent what everybody knows, or what is common knowledge, respectively. In this paper we study a combination of epistemic logic with linear temporal logic (LTL) called CKL_n [5].

Checking that a particular protocol satisfies certain desirable properties is a *verification* problem. One of the most successful general approaches to verification is known as *model checking* [2]. Model checking approaches to the analysis of security protocols have proved rather successful (for an early example, see the work of Dolev and Yao [3]). The basic idea here is to examine all possible execution traces by the protocol under investigation, in the presence of a malicious intruder with well-defined capabilities. One then either determines that the protocol enforces its security guarantees, or else obtains a sample trace of an attack on the protocol. This has led to the discovery of a number of attacks against existing protocols (for example, Lowe [8] used a model checker for CSP to find a previously unknown error in the Needham-Schroeder Public-Key Authentication protocol [10]). However we are not concerned with authentification but with secrecy, which requires a different approach.

One of the problems with using model checking to verify properties involving knowledge is that existing model checkers are designed to verify temporal, rather than epistemic properties. This is also addressed by Benerecetti [1], but for a different model checker than the one we are using. In this paper, we use and further develop ideas from a previous paper [11] to deal with epistemic model checking. We do this by considering the "Russian cards problem" [13], which will be explained in the next section. The setting differs from problems such as the bit transmission problem [9, 11] or the dining cryptographers [12], in the sense that we also give an account of an *adversarial* agent, as opposed to a *co-operative* agent. Section 2 introduces the logic of knowledge and time that we are using. In Section 4 we describe how we find solutions to this problem, and in Section 5 we present some conclusions.

2 The Logic of Knowledge and Time

Epistemic logic originates in philosophy [6] but has been successfully used in the domain of computer systems to capture the knowledge of agents in interpreted systems [4]. Epistemic logic alone is not sufficient for expressing changes or updates of knowledge, but the logic can be used as a basis for the definition of other logics. We define the language of epistemic logic in the following way, using a set op primitive

propositions Φ.

$\langle wff \rangle$	::=	**true**	/* logical constant for truth */
	|	any element of Φ	/* primitive propositions */
	|	$\neg\langle wff \rangle$	/* negation */
	|	$\langle wff \rangle \vee \langle wff \rangle$	/* disjunction */
	|	$K_i\langle wff \rangle$	/* $(i \in Ag)$ agent i knows */
	|	$C\langle wff \rangle$	/* It is common knowledge that */

Epistemic logic is thus an extension of propositional logic with knowledge operators $K_i\varphi$, which can be read as: "Agent i knows that φ". The operator $C\varphi$ expresses common knowledge between all agents of φ. Informally a fact φ is common knowledge if everyone knows φ, everyone knows that everyone knows that φ, and so on. Epistemic logic can be interpreted over Kripke [9] models of the form

$$M = \langle S, \{\sim_i\}, \pi \rangle$$

where S and π are as defined above, and $\sim_i$ is an equivalence relation, modeling what agent i considers to be indistinguishable states. We then can interpret formulas on pairs $\langle M, s \rangle$, where s is a state from S. If φ is true in such a pair, we write $\langle M, s \rangle \models \varphi$. Interpretation is according to the next definitions.

$\langle M, s \rangle \models p$	iff $\pi(p) =$ *true*
$\langle M, s \rangle \models \neg\varphi$	iff not $\langle M, s \rangle \models \varphi$
$\langle M, s \rangle \models \varphi \wedge \psi$	iff $\langle M, s \rangle \models \varphi$ and $\langle M, s \rangle \models \psi$
$\langle M, s \rangle \models K_i\varphi$	iff for all $t : s \sim_i t$ implies $\langle M, t \rangle \models \varphi$
$\langle M, s \rangle \models C\varphi$	iff for all $t : s \sim^C t$ implies $\langle M, t \rangle \models \varphi$

The relation $\sim^C$ used in the definition of the common knowledge operator is the transitive closure of the union of all $\sim_i$ relations. We say that a formula φ is true in a model M, and write $M \models \varphi$, if $\langle M, s \rangle \models \varphi$ for every state s in M.

In this paper we look at a language CKL_n, which combines epistemic logic and linear time temporal logic (LTL). The language LTL is, like epistemic logic, an extension of propositional logic. In the case of LTL there are two new operators, $\bigcirc\varphi$ and $\varphi \mathcal{U} \psi$ which allow for reasoning about the future. Informally $\bigcirc\varphi$ means that φ is true in the next state, whereas $\varphi \mathcal{U} \psi$ means that ψ is true at some time in the future, and until that time φ is true. The complete grammar for CKL_n is the following.

$\langle wff \rangle$	::=	**true**	/* logical constant for truth */
	|	any element of Φ	/* primitive propositions */
	|	$\neg\langle wff \rangle$	/* negation */
	|	$\langle wff \rangle \vee \langle wff \rangle$	/* disjunction */
	|	$\bigcirc\langle wff \rangle$	/* next */
	|	$\langle wff \rangle \mathcal{U} \langle wff \rangle$	/* until */
	|	$K_i\langle wff \rangle$	/* $(i \in Ag)$ agent i knows */
	|	$C\langle wff \rangle$	/* It is common knowledge in that */

$\langle \mathcal{I}, (r,u) \rangle$	$\models_{CKL_n}$	**true**		
$\langle \mathcal{I}, (r,u) \rangle$	$\models_{CKL_n}$	p	iff	$p \in \pi(r,u)$ (where $p \in \Phi$)
$\langle \mathcal{I}, (r,u) \rangle$	$\models_{CKL_n}$	$\neg\varphi$	iff	$\langle \mathcal{I}, (r,u) \rangle \not\models_{CKL_n} \varphi$
$\langle \mathcal{I}, (r,u) \rangle$	$\models_{CKL_n}$	$\varphi \vee \psi$	iff	$\langle \mathcal{I}, (r,u) \rangle \models_{CKL_n} \varphi$ or $\langle \mathcal{I}, (r,u) \rangle \models_{CKL_n} \psi$
$\langle \mathcal{I}, (r,u) \rangle$	$\models_{CKL_n}$	$K_i\varphi$	iff	for all (r',v) in $\mathcal{I}$, if $(r,u) \sim_i (r',v)$ then $\langle \mathcal{I}, (r',v) \rangle \models_{CKL_n} \varphi$
$\langle \mathcal{I}, (r,u) \rangle$	$\models_{CKL_n}$	$C\varphi$	iff	for all (r',v) in $\mathcal{I}$, if $(r,u) \sim_\Gamma^C (r',v)$ then $\langle \mathcal{I}, (r',v) \rangle \models_{CKL_n} \varphi$
$\langle \mathcal{I}, (r,u) \rangle$	$\models_{CKL_n}$	$\bigcirc\varphi$	iff	$\langle \mathcal{I}, (r,u+1) \rangle \models_{CKL_n} \varphi$
$\langle \mathcal{I}, (r,u) \rangle$	$\models_{CKL_n}$	$\varphi \mathcal{U} \psi$	iff	$\exists v \in I\!N$ s.t. $(u \leq v)$ and $\langle \mathcal{I}, (r,v) \rangle \models_{CKL_n} \psi$ and $\forall w \in \{u, \ldots, v-1\}$, we have $\langle \mathcal{I}, (r,w) \rangle \models_{CKL_n} \varphi$

Figure 1: Semantics of CKL_n

Two useful definitions are $\diamondsuit\varphi = \mathbf{true}\,\mathcal{U}\,\varphi$ and $\Box\varphi = \neg\diamondsuit\neg\varphi$. The intended reading of $\diamondsuit\varphi$ is that at some point in the future φ will happen. Its dual $\Box\varphi$ means that φ holds in all future states.

The semantics of CKL_n are given next. Let Ag denote the set of agents. We assume each agent $i \in Ag$ can be in any of a set L_i of local states. An agent's local state contains all the information required to completely characterise the state of the agent: the value of each of its local variables, together with the value of its program counter. In particular, the *information* available to an agent is determined by its local state. The state of a system at any moment can thus be characterised by a tuple $\langle l_1, \ldots, l_n \rangle$, where $l_i \in L_i$ is the local state of agent i at this moment. We let $G \subseteq L_1 \times \cdots \times L_n$ denote the *reachable global* states of the system (i.e., the set of states that a system may possibly enter during a legal computation sequence).

A *run* is a function $r : I\!N \rightarrow G$, which associates with every natural number $u \in I\!N$ a global state $r(u)$. The idea is that a run represents one possible computation of a system: in general, a system may have a number of possible runs, and so we say a *system* is a set of runs; we use $\mathcal{R}$ to denote a system. A run together with a time point is a *point*: a point (r,u) defines a global state $r(u)$. We denote the i'th component of the tuple $r(u)$ by $r_i(u)$. Thus $r_i(u)$ is the local state of agent i in run r at "time" u. A *model* or *interpreted system* for CKL_n is a pair $\mathcal{I} = \langle \mathcal{R}, \pi \rangle$, where $\mathcal{R}$ is a set of runs system and $\pi : \mathcal{R} \times I\!N \rightarrow 2^\Phi$ is a valuation function, which gives the set of primitive propositions true at each point in $\mathcal{R}$ [4, pp.110–111].

We define the semantics of CKL_n using a satisfaction relation "$\models_{CKL_n}$", which holds between pairs of the form $\langle \mathcal{I}, (r,u) \rangle$, (where $\mathcal{I}$ is an interpreted system and (r,u) is a point in $\mathcal{I}$), and formulas of CKL_n. We read $\langle I, (r,u) \rangle \models_{CKL_n} \varphi$ as "φ is satisfied (equivalently, is true) at point (r,u) in $\mathcal{I}$". The rules defining $\models_{CKL_n}$ are given in Figure 1.

We use $\mathcal{I} \models \varphi$ as a shorthand for saying that φ holds in the initial state of any run of $\mathcal{R}$: $\mathcal{I} \models \varphi \;\Leftrightarrow\; \forall s : \langle \mathcal{I}, (s,0) \rangle \models \varphi$.

2.1 Model Checking

Model checking was developed as a technique for automatically verifying that finite state systems satisfy requirements expressed in the language of temporal logic [2]. The term 'model checking' arises from the fact that the state transition system of a finite state system can be understood as a model (in fact, a Kripke structure) for temporal logic. Verifying that a system satisfies certain requirements, where these requirements are expressed in temporal logic, can then be understood as a problem of checking that the formula representing the requirements is realised in the model. In our work, we make use of the SPIN model checker for Linear-time Temporal Logic [7]. SPIN takes as input a system S described using the PROMELA modelling language, and a formula φ of LTL, and checks whether or not the formula is realised in the system; if the answer is "no", then it produces an execution trace of S that falsifies φ. While SPIN has been widely used for the verification of LTL properties, it is not currently equipped to deal with LTL formulas that contain knowledge modalities. In previous work [11], we explored the use of SPIN for model checking temporal knowledge properties of systems, by finding *local propositions* to stand for knowledge modalities in formulas.

LTL is a modal propositional logic which allows to specify properties of linear sequences of states — *runs*. Each PROMELA specification defines a set of runs, and SPIN checks whether the LTL formula holds for all runs. The PROMELA specification itself does not have to be linear (two different states can be reachable from the same state) but properties that must refer to two different runs containing the current state cannot be expressed in LTL.

3 The Russian Cards Problem

The Russian cards problem first appeared in the Russian mathematics olympiad in 2000 [13]. It has subsequently been picked up by logicians as an example of a information based security. The official informal description, translated from Russian, is the following.

> From a pack of seven known cards two players each draw three cards and a third player gets the remaining card. How can the players with three cards openly (publicly) inform each other about their cards, without the third player learning from any of their cards who holds it?

We call the players Alice, Bob and Caroline (also denoted A, B and C), and refer to the different cards as if they are numbered from 0 to 6. A solution for this problem is any dialogue between Alice and Bob which informs them of the current card deal but keeps Caroline in the dark about the location of any specific card. As an attempt to a solution of the Russian Cards problem, one might devise a very long and complicated dialogue. However, instead, we are looking for a *direct exchange solution*, in which the actual dialogue is short: Alice makes a single (true) statement from which Bob learns which cards she has. Bob can then deduce which card Caroline has and (truthfully) announce this. If Alice learns Caroline's card she can deduce which cards Bob must have. In all such solutions, it is sufficient and necessary that Bob's reply to Alice is to announce

which card Caroline holds. A direct exchange is thus completely characterised by Alice's initial announcement. Informally a statement is a *solution* if it informs Bob of the card deal and, after Alice announces this statement, Caroline does not know any of the cards.

In order to look at a few examples, assume that Alice has cards $0, 1, 2$, Bob has $3, 4, 5$ and Caroline holds card 6. It has been proven [13] that any statement that Alice can knowingly make is equivalent to a statement about her own cards, so we can limit our discussion to such statements.

Consider first the case where A says: I have cards $0, 1$ and 2, or I have $3, 4$ and 5. This informs Bob of Alice's cards and does not enable Caroline to learn who has which cards. Unfortunately Alice considers it possible that Caroline holds card 3, in which case should could not safely make this statement. Therefore this example is not a solution to the problem.

A second case is the statement by Alice: "I have $0, 1$ and 2 or I have neither of these cards". This looks better because even if Caroline has card 3, she cannot conclude from the truth of the statement that Alice holds any specific card. However, Caroline can reason in the following way: Suppose that Alice does not have cards $0, 1$ and 2. Then she should consider it possible that I have card 1. In that case I would learn from this statement that Bob has cards 0 and 2, which should not happen. Therefore, Alice would never make this statement if she did not hold $0, 1$ and 2. Since Caroline can draw this conclusion, this is thus also not a solution.

An example of a solution is that Alice says "I have one of the following five sets of three cards: 0, 1 and 2, 0,3 and 4, 1, 3 and 5, 2,3 and 6 or 4,5 and 6".

3.1 Reasoning about Russian Cards in Epistemic Logic

A logical description of this problem needs propositional logic to describe a deal of cards. From now on A, B and C will be used to denote the three agents. Moreover, i is a variable over the agents, and the variables x and y range over the deck of cards $\mathcal{D} = \{0, 1, 2, 3, 4, 5, 6\}$. The atoms a_x are used to denote that fact that A holds card x. Similarly for b_x and c_x with respect to B and C, respectively.

The fact that these propositions must represent a card deal puts constraints on these propositions. For instance, every card i must be held by exactly one person (∇ denotes exclusive or): $\bigwedge_{x\in\mathcal{D}}(a_x \nabla b_x \nabla c_x)$. Let us now denote the state in which A has cards 0,1,2, B has 3,4,5 and C has 6 by $s = 012|345|6$. In the sequel, we thus consider states s of the form $x_0x_1x_2|x_3x_4x_5|x_6$ where $x_0x_1x_2x_3x_4x_5x_6$ is a permutation of $\{0, \ldots, 6\}$ and where $x_0 < x_1 < x_2$ and $x_3 < x_4 < x_5$.

The ultimate goal of A's announcement is that B knows the deal of cards and C is ignorant about all cards. This is expressed in epistemic logic by the following formulas

$$\begin{array}{lcl} \mathsf{bknows} & = & \bigwedge_{i\in\mathcal{D}} a_i \rightarrow K_B a_i \\ \mathsf{cig} & = & \bigwedge_{i\in\mathcal{D}} (\neg K_C a_i \wedge \neg K_C b_i) \end{array}$$

What we have to check is thus that these formulas hold at the end of the protocol. Assume that end is some proposition that only holds at the end of each run. We must verify the following, in order to establish that φ is a solution. The model M_φ is the

model in which A uses statement φ.

$$M_\varphi \models \Box(\text{end} \implies (\mathsf{bknows} \wedge \mathsf{cig}))$$

4 Model Checking and the Russian Cards Problem

We now turn to the key contribution of this paper: the use of model checking to automatically verify properties of the Russian Cards problem. Before one can apply model checking to prove properties of any protocol, one must obtain a formal specification implementing the protocol. This is not trivial for the Russian Cards because the problem description defines the epistemic relations directly, while in most specification languages we must deal the cards explicitly by means of a new process. In PROMELA one can only define the dynamic transitions directly. Epistemic relations are derived by using the *interpreted systems* model of knowledge [4], in which each process only has access to its own variables.

For each process i, we define an equivalence relation $\sim_i$ between points as follows: $(s, u) \sim_i (t, v)$ iff all variables accessible to i have the same value in (s, u) and (t, v). For the purpose of knowledge, the program counter of each process, which is used to remember which is the next computation step, is also considered an accessible variable for the corresponding process. The definition of knowledge states that something is known if it is true in any reachable state that cannot be distinguished by the agent.

Modeling the Russian Cards Problem in PROMELA We have constructed the PROMELA specification as a message passing system in which all messages are stored in variables accessible to all agents that have received the message [4]. A standard binary representation for every formula has been used, ensuring that no information is contained in the way formulas are written. The decisions regarding the task of each module, the representation of cards and the representation of statements are explained below. Once these decisions have been made the implementation is fairly straightforward, and this gives us good reason to believe that the specification we use indeed implements the protocol we want to check.

The PROMELA specification we describe here is given in full in the appendix of this paper. It can also be downloaded from `www.csc.liv.ac.uk/~sieuwert/`.

A specification in SPIN consists of multiple processes that operate in parallel. Each agent in a multi agent system can be modeled as a separate process and this leads to the definition of three processes: `A()`, `B()` and `C()`. Every process `X()` has access to a variable x denoting its hands of cards. Based on their respective cards `A()` must make a statement and `B()` and `C()` must listen to and interpret the statement. That statement is stored in the variable `disj`, accessible to all three processes. Process `B()` moreover has a variable `ba` in which he stores his calculated hand of `A()`.

Another process is needed to generate a deal of cards, since no single agent can be said to be in control of the deal of cards. The process responsible for determining the deal of cards is called `deal()`. The next table shows all processes with their accessible variables.

process	variables
deal()	a,b,c
A()	a,disj
B()	b, ba, disj
C()	c,disj

In each run all processes perform the same steps but on a different deal of cards. The next table sums up what each process does. The labels `END` indicate the points in the execution of `B` and `C` where their knowledge must be tested. Naturally these labels are placed at the end of their specification since we are interested in the knowledge state after the announcement.

process	actions
`deal()`	set up a deal in `a,b` and `c` signal `A()`
`A()`	wait for signal from `deal()` choose φ and place in `disj` wait until φ is true signal `B()` , signal `C()`
`B()`	wait for signal from `A()` let `ba` be a hand from `disj` for which no card is both in `b` and `ba` `END:` do nothing
`C()`	wait for signal from `A()` `END:` do nothing

The specification code is intended to test the suitability for a specific formula φ. Process A checks whether φ, which is a statement about A's cards, is true. If not, it blocks. This may not seem realistic. In real life, the process would use another statement than φ and process B and C would continue as usual. However, the inclusion of other statements than φ in the model would make the number of points of the system much larger. Since we are only interested in runs in which φ holds, we have chosen to cut all other runs short. This way the runs in which we are not interested are not expanded, and we save valuable memory and computation time.

Capturing the Protocol Requirements We now consider the requirements that we wish to check. They are captured in CKL_n. The requirement can be broken down in two parts bknows and cig , which have to be tested at the end label of the corresponding process.

The formula bknows is an example of a positive knowledge requirement, which often occurs between cooperating agents. Because an explicit variable ba, that models what B may think that A has, has been supplied in the specification code, it is sufficient to test that the procedure which guesses the value of this variable is correct. This is expressed in the following LTL formula, which we have used.

$$\Box(\text{at}_B(\text{END}) \rightarrow ba = a) \tag{1}$$

An interesting question is how close the correspondence between checking bknows using formula 1 and the general method [11] available for model checking is. The general method asks the user to find *local propositions* which correspond to the knowledge formula. The method can then prove the suitability of these local formulas. Instead of coming up with a local formula for each card x, describing the local states in which a_x is true, we have introduced a procedure for determining the value of all propositions. We check the suitability of that procedure instead of checking local propositions. The method is thus similar except that we do more of the work in the specification instead of in the LTL formulas. This variant on the local propositions method is in our opinion acceptable in situations where one does not start with a given formal specification, but only with an informal protocol description. The advantage of the above formula is that by working in the specification one can avoid having to mention all the cards explicitly. Checking one procedure is much more elegant than checking a local proposition for each card.

To see that the truth of bknows indeed follows from the truth of formula 1, assume that the model checker cannot find any counterexample to the formula. This shows that the variables `a` and `ab` are equal in each end state of process `B()`. We have not specified any propositions referring to the variable `ba`, but it is not unreasonable to assume a set similar to the propositions referring to A's cards: $ba_0, ba_1, \ldots ba_6$. Using these propositions we can express the model checking result in LTL logic extended with knowledge operators. The model checking result can be expressed as $\mathcal{I} \models \Box(\text{at}_B(\text{END}) \rightarrow (a_x \leftrightarrow ba_x))$.

Everything that is true at all points of every run in the system I is commonly known by all agents, so in particular it is known by process `B()` in all points in all runs: $\mathcal{I} \models \Box K_B(\text{at}_B(\text{END}) \rightarrow K_B(a_x \leftrightarrow ba_x))$. Because knowledge distributes over implication this gives $\mathcal{I} \models \Box(K_B\text{at}_B(\text{END}) \rightarrow (K_B a_x \leftrightarrow K_B ba_x))$. The variable ba is part of `B()`'s local state, giving $\mathcal{I} \models \Box(ba_x \leftrightarrow K_B ba_x))$, and `B()` knows when it is in its end state, giving $\mathcal{I} \models \Box(\text{at}_B(\text{END}) \leftrightarrow K_B\text{at}_B(\text{END}))$. Since x was chosen arbitrarily, we have proven

$$\mathcal{I} \models \Box(\text{at}_B(\text{END}) \rightarrow \mathsf{bknows})$$

Checking the Absence of Knowledge So far, we have been concerned with establishing the *presence* of knowledge — that after the public announcement by A is made, B knows the deal. But we also need to show the *absence* of knowledge, i.e., that C does not know the deal after the announcement. This is more problematic than establishing the presence of knowledge. One can show that a certain piece of knowledge is not present by attempting to verify that the knowledge is present [11]. This verification will fail, and result in a counterexample. The counterexample shows that at a certain point the statement φ is not necessarily true, so it cannot be known ($K_x\varphi$ does not hold at point (s, u)).

But more often, one wants to prove the absence of knowledge in a set of points S instead of just a single point (s, u). All states in the set can for example correspond to the same time point for a certain agent in the protocol, but have different values for the variables. A suitable definition of truth in a set of points is that the formula must be true at all points in the set: $(\mathcal{I}, S) \models \varphi \quad \Leftrightarrow \quad \forall(s, u) \in S \ \langle \mathcal{I}, (s, u)\rangle \models \varphi$. The set

S can for instance correspond to all points in which agent x is at a certain label in its specification.

One example of an absence of knowledge formula is $\neg K_C a_0$. This formula states that C does not know that A has card 0. The entire statement cig is a conjunction of 14 statements like this. The statement must hold in the end state of any run. Let S be the set of points in which C has reached their end state. One way to prove absence of knowledge of a formula in a point is to show that there is at least one indistinguishable point at which the formula is not true. In the example, one can do this by letting SPIN verify that a_0 is true in all states in S. If the verification fails, a counterexample run r will result, containing a point (s, u) which is in S but in which a_0 is not true. This tells us $\neg\forall(s, u) \in S\ \langle \mathcal{I}, (s, u)\rangle \models a_0$ One might call this *weak* absence of knowledge, because it shows one $(s, u) \in S$ in which a_0 is not true, i.e., in the local state $r_C(s(u))$ of C corresponding to s the formula is thus not known. If agent C has a unique local state $r_C(s(u))$ in all points of set S this is indeed a proof that C does not know a_0. But if C has more than one local state this is not a complete proof. It is not the same thing as $(\mathcal{I}, (s, u)) \models \neg K_C a_0$, which we might call *strong* absence of knowledge.

From the definition of knowledge in a set of points one might get the impression that one needs a counterexample for each member of the set. We obtain the following.

$$\begin{aligned} (\mathcal{I}, S) \models \neg K_x\varphi \quad &\Leftrightarrow \\ \forall(s, u) \in S\ \langle \mathcal{I}, (s, u)\rangle \models \neg K_x\varphi) \end{aligned} \tag{2}$$

In formula 2 containing universal quantification one can do an expansion based on formula 3, which is derived from the definition of knowledge and negation.

$$\begin{aligned} \langle \mathcal{I}, (s, u)\rangle \models \neg K_x\varphi \quad &\Leftrightarrow \\ \exists(t, v)((s, u) \sim_x (t, v) \wedge \langle \mathcal{I}, (t, v)\rangle \models \neg\varphi) \end{aligned} \tag{3}$$

We see that proving $\neg K_x\varphi$ on a set of points S involves finding a counterexample (t, v) for each point (s, u) in S. However, if $(s, u) \sim_x (s', u')$ and both are in S, then the same point (t, v) is also a counterexample for (s', u'). This is true because $(s, u) \sim_x (s', u')$ and $(s, u) \sim_x (t, v)$ implies $(s', u') \sim_x (t, v)$. It is sufficient to find one counterexample (t, v) for each different local state of the agent x. The number of local states that agent x might have at points in S will often be much smaller than S itself. Agent C for instance has only 7 different end states, one for each card.

The agent for which we want to prove absence of knowledge is agent C. The set of points S consists of all the points in which agent C reaches its END label. On this set C is required to have no knowledge of all propositions a_i and b_i.

$$\mathsf{cig} = \bigwedge_{i\in\mathcal{D}} (\neg K_C a_i \wedge \neg K_C b_i)$$

SPIN is much slower at verifying large formulas, therefore we verify this requirement by verifying each of the fourteen conjuncts separately. In the following paragraphs the steps needed for verification of $\neg K_C a_0$ are explained.

The property must hold at the end of each run, when process `C()` is at the label `END`. At this label, C has seven different states, because it can have seven different cards. In each of these states a different proposition from the set $T = \{c_0, c_1, c_2, c_3, c_4, c_5, c_6\}$ is true. It could be the case that some of these local states are unreachable. For instance, if $\varphi = a_0$ then c_0 will always be false. If checking the formula $\Box\neg c_x$ does not result in a counterexample run, the state c_x should be removed from set T. In most cases, and also for the formula φ that is present in the program listing all of these local states are reachable.

For all reachable local states one must find a counterexample run, in which the agent is in the local state but the formula is not true. These counterexamples can be found by using SPIN to check and disprove the following formulas, for all $i \in \mathcal{D}$:

$$\Box((\text{at}_C(\text{END}) \wedge c_i) \rightarrow a_0)$$

Finding these seven counterexamples takes seven runs of SPIN and proves C's ignorance of the fact that A holds card 0. A complete proof of all fourteen conjuncts would require 98 runs of SPIN. The number of runs depends on the number of local states and one needs a set of propositions enumerating all local states in order to write down the requirement.

5 Conclusions

It is possible to use model checking to prove knowledge properties of protocol specification, even using a standard model checker such as SPIN. The method used is less efficient when proving absence of knowledge for a set of states, since the number of times one needs to run SPIN depends on the number of local states of the ignorant agent; and the number of states will of course be exponential in the number of bits that make up that agent's local state.

One important consequence is that *adversarial* multi agent systems are more difficult to model check, because establishing the absence of knowledge typically arises in the adversarial case. In *co-operative* agent systems one can sometimes get away with the easier to prove notion of *weak absence of knowledge*.

In future work, we intend to refine our techniques to develop a general methodology for model checking knowledge, and we intend to apply this methodology to larger and more realistic scenarios.

Appendix

In this appendix we present the PROMELA source code for the Russian Cards problem.

Spin models usually start with some useful macros. We have chosen to use a binary representation of hands of cards. Each bit position represents whether the associated card is present. The next macros use this representation. The macro `overlap` returns true if its arguments have any card in common. `noverlap` is the negation of `overlap`. The macro `card` returns the bit pattern of each single card.

```
#define card(j)        (1<<(j))
#define overlap(i,j)   (((i)&(j))!=0)
#define noverlap(x,y)  (((x)&(y))==0)
```

All variables are declared to be global, since local variables are not sharable between processes and it is not possible to refer to them in CKL_n formulas. The variables `a,b,c` represent the hand of each agent. `ba` represents what B thinks that A has.

```
byte disj[7];
byte a=0,b=0,c=0; ba=0;
```

Next we declare a few communication channels that are used for synchronisation between the agents.

```
chan dealtoa=[0] of{byte};
chan atoc=[0] of{byte};
chan atob=[0] of{byte};
```

These channels are suitable for passing bytes. The values that are sent through these channels are not significant. Channel $x$to$y$ is used for synchronising between process x and process y: x sends a number and y will wait for that number. `Deal()` is the only process that does not have to wait for other processes so it is the first one to do something. It puts cards in the variables `a`, `b` and `c`. In the code below a card `x` is chosen and assigned to any player that has not received its maximal number of cards yet. The counters `na`,`nb` and `nc` are used to keep track of the number of cards each player has.

```
active proctype deal(){
 byte na=0,nb=0,nc=0; byte x;
 do
 ::na+nb+nc==7 ->break
 ::na+nb+nc<7 ->
  if
  ::noverlap(a|b|c,card(0))->x=card(0)
   ...
  ::noverlap(a|b|c,card(6))->x=card(6)
  fi;
  //select someone to give the card to.
  if
  :: na<3 -> a=a+x; na++
  :: nb<3 -> b=b+x; nb++
  :: nc<1 -> c=c+x; nc++
  fi
 od;
 dealtoa!a;
}
```

Process `A()` waits for a signal from process `deal()` and then it sets a statement in `disj`. The code shown writes the formula $((a_0 \wedge a_1 \wedge a_2) \vee (a_0 \wedge a_3 \wedge a_4) \vee (a_1 \wedge a_3 \wedge a_5) \vee (a_4 \wedge a_5 \wedge a_6) \vee (a_2 \wedge a_3 \wedge a_6))$. Only one formula φ can be tested at the same time. After writing the formula process `A()` waits until its cards are equal to

one of the possibilities in the formula. The process will either continue immediately, or block forever. As discussed in the main paper, this allows us to study only part of all possible runs.

```
active proctype A(){
 dealtoa?a;
 disj[0]=card(0)+card(1)+card(2);
 disj[1]=card(0)+card(3)+card(4);
 disj[2]=card(1)+card(3)+card(5);
 disj[3]=card(4)+card(5)+card(6);
 disj[4]=card(2)+card(3)+card(6);
 //only proceed if formula is true
 a==disj[0]||a==disj[1]||a==disj[2]||
 a==disj[3]||a==disj[4]||a==disj[5]||
 a==disj[6];
 atob!0;
 atoc!0;
}
```

Agent B waits until A signals that it has made a statement. Then, B will non-deterministically choose a hand of cards for agent A that is consistent with its own cards. After this choice it reaches the statement skip with the label END attached to it.

```
active proctype B(){
 atob?0;
 if
 ::disj[0]!=0&&noverlap(disj[0],b)
  -> ba=disj[0]
  ...
 ::disj[6]!=0&&noverlap(disj[6],b)
  -> ba=disj[6]
 fi;
 END:skip
}
```

Agent C waits for a signal from agent A and reaches its `END` state. There is nothing to do for this agent because the knowledge properties on this agent refer to the knowledge of this agent according to the given definition. The knowledge of for instance A's cards does not have to be explicitly available inside this process.

```
active proctype C(){
 atoc?0; //check if A is ready
 END:skip //here the properties must hold
}
```

The CKL_n formulas that are evaluated use propositions to express basic facts. These propositions are defined in the following macro's and correspond to the propositions in the logical formalisation of section 3.

```
#define c_end C[3]@END
#define b_end B[2]@END
```

```
#define a_0 ((a&1)!=0)
#define a_1 ((a&2)!=0)
...
#define c_5 ((c&32)!=0)
#define c_6 ((c&64)!=0)
```

References

[1] M. Benerecetti and A. Cimatti. Symbolic model checking for multi agent systems. In *Proceedings of the ICLP'01 workshop on Computational Logic in Multi-Agent Systems*, 2001.

[2] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press: Cambridge, MA, 2000.

[3] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1998.

[4] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning about knowledge*. The MIT Press: Cambridge, MA, 1995.

[5] J. Y. Halpern and M. Y. Vardi. The complexity of reasoning about knowledge and time. I. Lower bounds. *Journal of Computer and System Sciences*, 38:195–237, 1989.

[6] J. Hintikka. *Knowledge and Belief: an introduction to the logic of the two notions*. Cornell University Press: Ithaca, NY, 1962.

[7] G. Holzmann. The model checker Spin. *IEEE Trans. on Software Engineering*, 23:279–295, May 1997.

[8] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FKR. In *Tools and Algorithms for the Construction and Analysis of Systems (TACS)*, volume 1055 of *LNCS*, pages 147–166. Springer-Verlag, 1996.

[9] J.-J. Ch. Meyer and W. van der Hoek. *Epistemic Logic for AI and Computer Science*. Cambridge University Press: Cambridge, England, 1995.

[10] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communcations of the ACM*, 21(12):993–999, 1978.

[11] W. van der Hoek and M. Wooldridge. Model checking knowledge and time. In D. Bošnački and S. Leue, editors, *Model Checking Software, Proceedings of SPIN 2002 (LNCS Volume 2318)*, pages 95–111. Springer-Verlag: Berlin, Germany, 2002.

[12] R. van der Meyden and K. Su. Symbolic model checking the knowledge of the dining cryptographers. under review, 2004.

[13] H. P. van Ditmarsch. The russian cards problem. *Studia Logica*, 75(4):31–62, 2003.