Symbolic Model Checking the Knowledge of the Dining Cryptographers

Ron van der Meyden*
National ICT Australia &
School of Computer Science and Engineering
University of New South Wales, Australia
meyden@cse.unsw.edu.au

Kaile Su†
Department of Computer Science
Zhongshan University, China
issk1s@zsu.edu.cn.

Abstract

This paper describes the application of symbolic techniques (in particular, OBDD's) to model checking specifications in the logic of knowledge for an agent operating with synchronous perfect recall *in an environment of which it has incomplete knowledge. It discusses the application of these techiques to the verification of a security protocol: Chaum's Dining Cryptographers protocol, which provides a mechanism for anonymous broadcast.*

1. Introduction

Model checking, now well established for temporal verification of finite state systems, has in recent years begun to be applied to the verification of security protocols [24, 20, 8]. One of the key concerns in security protocols is the need to model and analyze the way information flows within the protocol. Expressing information flow is a strength of the logic of knowledge [13, 10], and numerous works, starting with [5], have argued that security specifications should be expressed in an epistemic formalism. However, relatively little work has been done on the application of model checking knowledge in security protocols – in part because there have been no model checkers for the logic of knowledge. In this paper we take some initial steps to address this deficit.

The application of model checking within the context of the logic of knowledge was first mooted by Halpern and Vardi [14]. A number of algorithms for model checking epistemic specifications and the computational complexity of the related problems are studied in [22, 23]. In this paper, we begin the work of transforming the theoretical results in the latter works into a practical implementation, by describing how symbolic techniques (in particular, OBDD's [4]) may be used to implement one of the algorithms of [22]. In particular, we show how these techniques may be applied to specifications concerning the knowledge of a single agent operating with *synchronous perfect recall* in an environment of which it has incomplete knowledge. An agent has perfect recall if it remembers all the information it has acquired over time; synchrony adds the assumption that the agent has access to the global clock, so that it always knows the time.

To illustrate the utility of model checking this class of specifications with respect to these assumptions, we consider a protocol drawn from the security literature: Chaum's Dining Cryptographers protocol [6], which forms the basis of Herbivore, a recently implemented anonymous broadcast system [11]. The Dining Cryptographers protocol has a substantially different character to others that have been considered in the literature on model checking security protocols, in that it exploits some quite specific properties of one-time pads rather than relying on generic properties of encryption. The specification of this protocol also inherently concerns what the agents in the system know, rather than what they are able to achieve. The specification of the protocol requires the agents to know certain information after running the protocol, but also requires that they do not know certain other information. Because of the information theoretic nature of this specification, approaches to model checking security protocols based on temporal logic and a Dolev-Yao style modelling of information flow are unable to verify this protocol. Moreover, our approach yields a guarantee of correctness, whereas Dolev-Yao based approaches to model checking security specifications are sound but not complete, in that they potentially validate protocols containing errors. We justify these claims in detail in Section 5.

The structure of the paper is as follows. In Section 2, we present a formal model for the environments in which agents operate, and describe how an environment, together

* National ICT Australia is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council. Work supported by an ARC Discovery grant.

† Work performed while a visiting fellow at UNSW. Also partially supported by the National Natural Science Foundation of China under Grant 60073056.

with a protocol for each agent in the system, generates a semantic structure supporting the definition of the knowledge of the agents in the system. Section 3 recalls an algorithm from [22], which computes a data structure that may be used to verify properties of an agent's knowledge. This section also explains how symbolic representations may be applied to implement this algorithm. Section 4 introduces the Dining Cryptographers protocol, and describes a system we have developed based on the results of the preceding sections, and how it has been applied to this protocol. Section 5 concludes by discussing related literature on security protocol verification.

2. Knowledge in a Finite Environment

In this section we define the semantic framework within which we study the model checking of specifications in the logic of knowledge. The intention of this framework is to capture our intuitions about the way that states of information arise in a computational setting. While intuitive, it is not well suited to a direct implementation of model checking. In the next section we develop an approach to translate the framework of the present section to one suited to algorithmic verification.

Systems will be decomposed in our framework into two components: the program, or *protocol* being run, and the remainder of the system, which we call the *environment* within which this protocol operates. We begin by presenting a semantic model, from [21], for the environment. This model is an adaptation of the notion of *context* of Fagin et al. [10].

Intuitively, we model the environment as a finite-state transition system, with the transitions labelled by the agents' actions. For each agent $i = 1 \ldots n$ let ACT_i be a set of *actions* associated with agent i. In order to model nondeterminism, we will also consider the environment to be able to perform actions, so assume additionally a set ACT_e of actions for the environment. A *joint action* will consist of an action for each agent and an action for the environment, i.e., the set of joint actions is the cartesian product $ACT = ACT_e \times ACT_1 \times \ldots \times ACT_n$.

Suppose we are given such a set of actions, together with a set of $Prop$ of atomic propositions. Define a *finite interpreted environment for n agents* to be a tuple E of the form $\langle S_e, I_e, P_e, \tau, O_1, \ldots, O_n, \pi_e \rangle$ where the components are as follows:

1. S_e is a finite set of *states of the environment*. Intuitively, states of the environment may encode such information as the state of communications channels, outcomes of coin-flips, the values of local variables maintained by the agents, program counters for the agents, etc.
2. I_e is a subset of S_e, representing the possible *initial states* of the environment.
3. $P_e : S_e \to \mathcal{P}(ACT_e)$ is a function, called the *protocol of the environment,* mapping states to subsets of the set ACT_e of actions performable by the environment. Intuitively, $P_e(s)$ represents the set of actions of the environment that are enabled when the system is in state s. We assume that this set is nonempty for all $s \in S_e$.
4. τ is a function mapping joint actions $\mathbf{a} \in ACT$ to state transition functions $\tau(\mathbf{a}) : S_e \to S_e$. Intuitively, when the joint action $\mathbf{a}$ is performed in the state s, the resulting state of the environment is $\tau(\mathbf{a})(s)$.
5. For each $i = 1 \ldots n$, the component O_i is a function, called the *observation function of agent i,* mapping the set of states S_e to some set $\mathcal{O}$. If s is a state in S_e then $O_i(s)$ will be called the *observation* of agent i in the state s.
6. $\pi_e : S_e \to \{0,1\}^{Prop}$ is an *interpretation*, mapping each state to an assignment of truth values to the atomic propositions in $Prop$.

A *run* r of an environment E is an *infinite* sequence $s_0 s_1, \ldots$ of states such that $s_0 \in I_e$ and for all $m \geq 0$ there exists a joint action $\mathbf{a} = \langle a_e, a_1, \ldots, a_n \rangle$ such that $a_e \in P_e(s_m)$ and $s_{m+1} = \tau(\mathbf{a})(s_m)$. For $m \geq 0$ we write $r_e(m)$ for s_m. For $k \leq m$ we also write $r[k..m]$ for the sequence $s_k \ldots s_m$. A *point* is a tuple (r, m), where r is a run and m a natural number. Intuitively, a point identifies a particular instant of time in the history described by the run.

Following [10], define a *system* to be a set $\mathcal{R}$ of runs and an *interpreted system* to be a tuple $\mathcal{I} = (\mathcal{R}, \pi)$ consisting of a system $\mathcal{R}$ together with an interpretation function π mapping the points of runs in $\mathcal{R}$ to assignments of truth value to the propositions in $Prop$.

The interpreted systems we deal with in this paper will have all runs drawn from the same environment, and the interpretation π derived from the interpretation of the environment by means of the equation $\pi(r,m)(p) = \pi_e(r_e(m))(p)$, where (r, m) is a point and p an atomic proposition. That is, the value of a proposition at a point of a run is determined from the state of the environment at that point, as described by the environment generating the run.

Environments, as defined above, admit a number of reasonable definitions of knowledge, depending on choices concerning the how agents make use of their observations in order to calculate what they know. In this paper, we will work with respect to a *synchronous perfect-recall* semantics of knowledge. Intuitively, this semantics amounts to assuming that agents have access to the global clock, and that they make optimal use of their observations, by taking all observations made in the past into account when computing what they know.[1] The synchronous perfect recall semantics is particularly apt in security protocols, such as the Dining Cryptographers protocol, for which the specification requires the lack of certain knowledge: ideally, we would like the proscribed knowledge to be absent, even if the agents make *optimal* use of the information they acquire.

The synchronous perfect recall assumption is formalized as follows. Given a run $r = s_0 s_1 \ldots$ of an environment with observation functions O_i, we define the *synchrononous perfect recall local state of agent i at time $m \geq 0$* to be the sequence $r_i(m) = O_i(s_0) \ldots O_i(s_m)$. That is, the synchronous perfect recall local state of an agent at a point in a run consists of a complete record of the observations the agent has made up to that point. (Since the synchronous perfect recall semantics is the only one we discuss in this paper, we will in the sequel write simply *local state* for the synchronous perfect recall local state.)

The local states may be used to define for each agent i a relation $\sim_i$ of *indistinguishability* on points, by $(r, m) \sim_i (r', m')$ if $r_i(m) = r'_i(m')$. Intuititively, when $(r, m) \sim_i (r', m')$, agent i has failed to receive enough information to time m in run r and time m' in run r' to determine whether it is in one situation or the other. Clearly, each $\sim_i$ is an equivalence relation. The use of the term "synchronous" above is due to the fact that an agent is able to determine the time simply by counting the number of observations in its local state. This is reflected in the fact that if $(r, m) \sim_i (r', m')$, we must have $m = m'$.

To specify systems, we will use a propositional multimodal language for knowledge and linear time based on a set *Prop* of atomic propositions, with formulae generated by the modalities $\bigcirc$ (next time), U (until), and a knowledge operator K_i for each agent $i = 1 \ldots n$. (Inclusion of the until operator is mostly for purposes of discussion). More precisely, the set of formulae of the language is defined as follows: each atomic proposition $p \in Prop$ is a formula, and if φ and ψ are formulae, then so are $\neg\varphi$, $\varphi \wedge \psi$, $\bigcirc\varphi$, $\varphi \; U \; \psi$, $K_i\varphi$ for each $i = 1 \ldots n$. When S is a set of modal operators, we write $\mathcal{L}_S$ for the fragment of this language based on the operators in S.

The semantics of this language is defined as follows. Suppose we are given an interpreted system $\mathcal{I} = (\mathcal{R}, \pi)$, where $\mathcal{R}$ is a set of runs of environment E and π is determined from the environment as described above. We define satisfaction of a formula φ at a point (r, m) of a run r in $\mathcal{R}$, denoted $\mathcal{I}, (r, m) \models \varphi$, inductively on the structure of φ. The cases for the temporal fragment of the language are standard:

1. $\mathcal{I}, (r, m) \models p$, where p is an atomic proposition, if $\pi(r, m)(p) = \texttt{true}$,
2. $\mathcal{I}, (r, m) \models \varphi_1 \wedge \varphi_2$, if $\mathcal{I}, (r, m) \models \varphi_1$ and $\mathcal{I}, (r, m) \models \varphi_2$,
3. $\mathcal{I}, (r, m) \models \neg\varphi$, if not $\mathcal{I}, (r, m) \models \varphi$,
4. $\mathcal{I}, (r, m) \models \bigcirc\varphi$, if $\mathcal{I}, (r, m + 1) \models \varphi$,
5. $\mathcal{I}, (r, m) \models \varphi_1 \; U \; \varphi_2$, if there exists $k \geq m$ such that $\mathcal{I}, (r, k) \models \varphi_2$ and $\mathcal{I}, (r, l) \models \varphi_1$ for all l with $m \leq l < k$.

The semantics of the knowledge operators is defined by

6. $\mathcal{I}, (r, m) \models K_i\varphi$, if $\mathcal{I}, (r', m') \models \varphi$ for all points (r', m') of $\mathcal{I}$ satisfying $(r', m') \sim_i (r, m)$

That is, an agent knows a formula to be true if this formula holds at all points that it is unable to distinguish from the actual point. This definition follows the general framework for the semantics of knowledge proposed by Halpern and Moses [13].

The systems $\mathcal{I}$ we will be interested in will not have completely arbitrary sets of runs, but rather will have sets of runs that arise from the agents running some program, or protocol, within a given environment. Intuitively, an agent's actions in such a program should depend on the information it has been able to obtain about the environment, but no more. We have used observations to model the agent's source of information about the environment. The maximum information that an agent has about the environment at a point (r, m) is given by the local state $r_i(m)$. Thus, it is natural to model an agent's program as associating to each local state of the agent the set of actions enabled when the agent is in that local state. We define a *protocol* for agent i to be a function $P_i : \mathcal{O}^+ \rightarrow \mathcal{P}(ACT_i)$. A *regular joint protocol* $\mathbf{P}$ is a tuple $\langle P_1, \ldots, P_n \rangle$, where each P_i is a (regular) protocol for agent i. If r is a run, we call $P_e(r_e(m)) \times P_1(r_1(m)) \times \ldots \times P_n(r_n(m))$ the *set of joint actions enabled by* $\mathbf{P}$ *at the point* (r, m).

The systems we consider will consist of all the runs in which at each point of time each agent behaves as required by its protocol. As usual, we also require that the environment follows its own protocol. Formally, the *system generated by a joint protocol* $\mathbf{P}$ *in environment* E is the set $\mathcal{R}(\mathbf{P}, E)$ of all runs r of E such that for all $m \geq 0$ we have $r_e(m + 1) = \tau(\mathbf{a})(r_e(m))$, where $\mathbf{a}$ is a joint action enabled by $\mathbf{P}$ at the point (r, m). The *interpreted system generated by a joint protocol* $\mathbf{P}$ *in environment* E is the interpreted system $\mathcal{I}(\mathbf{P}, E) = (\mathcal{R}(\mathbf{P}, E), \pi)$, where π is the interpretation derived from the environment E as described above.

To consider model cheking, we require a finite input. We have already constrained the environments we consider to be finite state. We now introduce some definitions that restrict the set of protocols to be finitely representable. A

1 Other reasonable assumptions to make about agents' knowledge would be that it is based just on the current observation, that it is based on the current observation plus the global clock value, or that it is based on the sequence of observations made by the agent, with stuttering eliminated.

protocol P_i is *regular* if for each $S \in \mathcal{P}(ACT_i)$, the set $P_i^{-1}(S)$ is regular. In this case, P_i can be represented as a deterministic finite state (Moore) automaton with inputs $\mathcal{O}$ and with outputs $\mathcal{P}(ACT_i)$, such that the automaton gives output $P_i(r_i(m))$ in the state reached when the automaton is given input string $r_i(m)$. A *regular joint protocol* $\mathbf{P}$ is a tuple $\langle P_1, \ldots, P_n \rangle$, where each P_i is a (regular) protocol for agent i. A protocol P_i (joint protocol $\mathbf{P}$) is *Markov* if $P_i(r_i(m))$ (respectively, $\mathbf{P}(r, m)$) depends only on the current state of the environment, i.e., $r_e(m)$. Given a regular joint protocol $\mathbf{P}$ and a finite environment E, we can (by adding information about the states of the automata representing $\mathbf{P}$ to the states of E) construct a finite state environment E' and a Markov joint protocol $\mathbf{P}'$ such that $\mathcal{I}(\mathbf{P}, E)$ and $\mathcal{I}(\mathbf{P}', E')$ are isomorphic. Consequently, we will, without loss of generality, assume that $\mathbf{P}$ is Markov in the next section.

3. Model Checking Knowledge Formulas

We now define the model checking problems we consider, describe previously developed algorithms solving these problems, and discuss how to use ordered binary decision diagrams to implement these algorithms.

We begin by considering specifications in $\mathcal{L}_{\{K_1,\ldots,K_n\}}$. The *alternation depth* of a formula of $\mathcal{L}_{\{K_1,\ldots,K_n\}}$ is the maximum number of alternations of *distinct* knowledge operators in the formula. For example, the formula p has alternation depth 0, and the formula $K_2K_1(p \Rightarrow K_1K_2q)$ has alternation depth 3 (note that the "consecutive" repetition of K_1 is counted only once, since this is not considered an alternation).

The first model checking problem we consider is a type of local model checking: testing if a formula is satisfied at a particular point (r, n) in the system generated by running a joint protocol in a given environment. In order to formulate this as a model checking problem, we need a finite input to the model checker. We have already restricted environments and protocols to be finitely representable. Clearly, the run r of the point at which we wish to verify a formula is not finite. To handle this, we proceed as follows.

Define a *trace* ρ of an environment E to be a finite prefix of a run of E. We write *traces*$(\mathbf{P}, \mathcal{E})$ for the set of finite prefixes of runs in $\mathcal{R}(\mathbf{P}, E)$. The following is shown in [22]: for formulas $\varphi \in \mathcal{L}_{\{K_1,\ldots,K_n\}}$, if $r[0 \ldots m] = r'[0 \ldots m]$, then $\mathcal{I}(\mathbf{P}, E), (r, m) \models \varphi$ iff $\mathcal{I}(\mathbf{P}, E), (r', m) \models \varphi$. That is, the truth value of such a formula at a point (r, m) depends only on the prefix of r to time m. (Clearly, this equivalence does not hold for formulas containing temporal operators.) This fact allows us to define the notion of the satisfaction of a formula $\varphi \in \mathcal{L}_{\{K_1,\ldots,K_n\}}$ at a trace ρ, by $\mathcal{I}(\mathbf{P}, E), \rho \models \varphi$ if $\mathcal{I}(\mathbf{P}, E), (r, m) \models \varphi$ for some (equivalently, every) run r such that $\rho = r[0 \ldots m]$.

We can now define the problem of model checking at a trace: given a finite environment E, a joint protocol $\mathbf{P}$ represented as a finite state automaton, a trace ρ of $\mathcal{I}(\mathbf{P}, E)$ and a formula $\varphi \in \mathcal{L}_{\{K_1,\ldots,K_n\}}$, determine if $\mathcal{I}(\mathbf{P}, E), \rho \models \varphi$. Define the natural number $C_k(\mathbf{P}, E)$ for a natural number k, finite state automaton representation $\mathbf{P}$ of a joint protocol and environment E for n agents by the recursion $C_0(\mathbf{P}, E) = 2n(|\mathbf{P}| + |E|)$, and $C_{k+1}(\mathbf{P}, E) = 2n(|\mathbf{P}| + |E|) \cdot 2^{C_k(\mathbf{P}, E)}$.

Theorem 3.1: *[van der Meyden [22]] Let E be a finite environment and $\mathbf{P}$ a finite state automaton representation of a joint protocol. For formulae $\varphi \in \mathcal{L}_{\{K_1,\ldots,K_n\}}$ of alternation depth bounded by k, and $\rho \in$ traces$(\mathbf{P}, E)$, $\mathcal{I}(\mathbf{P}, E), \rho \models \varphi$ can be decided in time $O(C_k(\mathbf{P}, E)) \cdot (|\rho| + |\varphi|))$.*

That is, the local model checking problem we have defined can be solved in time that is linear in the length of the trace and the size of the formula, but non-elementary in the alternation depth of the formula.

In what follows, we focus on a special case of this result, viz., formulas in $\mathcal{L}_{\{K_i\}}$. That is, we consider formulas in which all knowledge operators concern agent i only, so that we do not have nestings such as in K_1K_2p. All such formulas have alternation depth one (indeed it may be shown that each such formula is equivalent to one with no nesting of the operator K_i). The proof of Theorem 3.1 is easily explained in this case. Given trace $\rho = s_0 \ldots s_m$ of $\mathcal{R}(\mathbf{P}, E)$, define the set

$$S_i(\rho) = \left\{ r_e(m) \;\middle|\; \begin{array}{l} r \in \mathcal{R}(\mathbf{P}, E) \text{ and} \\ r_i(m) = O_i(s_0) \ldots O_i(s_m) \end{array} \right\}$$

consisting of states of the environment. Intuitively, $S_i(\rho)$ is the set of states that agent i (with synchronous perfect recall) considers possible at point (r, m) of any run r with $r[0 \ldots m] = \rho$.

Let π be the interpretation function of the environment E. Given a set of states S of the environment E, define the Kripke structure $M(S) = (S, S \times S, \pi)$, in which the set of worlds is S and the accessibility relation corresponding to the operator K_i is the universal relation on S. Satisfaction of a formula $\varphi \in \mathcal{L}_{\{K_i\}}$ at a world in $M(S)$ can be defined in the usual way. Write *fin*(ρ) for the final state of the trace ρ.

Lemma 3.2 : *For a formula $\varphi \in \mathcal{L}_{\{K_i\}}$, we have $\mathcal{I}(\mathbf{P}, E), \rho \models \varphi$ iff $M(S_i(\rho)), fin(\rho) \models \varphi$.*

This result enables us to determine the satisfaction of φ at a trace by means of a calculation involving $S_i(\rho)$. The following result provides an incremental way to compute this set. For this result, define the function $G : \mathcal{P}(S) \times S \rightarrow \mathcal{P}(S)$ by letting $G(X, s)$ be the set of states t' such that

1. there exists a state $t \in X$ and a joint action $\mathbf{a}$ enabled by $\mathbf{P}$ at t, such that $t' = \tau(\mathbf{a})(t)$ and

2. $O_i(t') = O_i(s)$.

(Recall that we assume that $\mathbf{P}$ is Markov.)

Lemma 3.3:

1. *If s is a state in the set I of initial states of E then $S_i(s) = \{t \in I \mid O_i(t) = O_i(s)\}$.*
2. *Suppose ρs is a trace with prefix ρ followed by final state s. Then $S_i(\rho s) = G(S_i(\rho), s)$.*

Thus, to verify $\mathcal{I}(\mathbf{P}, E), \rho \models \varphi$, it suffices first to compute $S_i(\rho)$ using Lemma 3.3, and then to check that $M(S_i(\rho)), \mathit{fin}(\rho) \models \varphi$. The first step can be done in time linear in the length of ρ, and the second can be done in time linear in the size of the formula. As presented, the computation needs sufficient space to represent a subset of the set of states of the environment. As the number of agents in the system increases, this can become large. Suppose, for example that we have n agents, each of which flips a coin (the dining cryptographers protocol considered in the next section is an example of this situation). Then the number of states of the environment is at least 2^n.

However, a variant of well known symbolic techniques can be used to implement this computation. Instead of explicitly listing the elements of $S_i(\rho)$, if we represent states by propositional assignments, then we can represent a set of states using an (ordered) binary decision diagram [4], or *OBDD*. OBDD's are space efficient representations of boolean functions of several variables. Ordering the input variables of a function $f : \{0,1\}^n \to \{0,1\}$ as $x_1, \ldots x_n$, we can represent the function as a full binary tree of height n in which the edges at level i are labelled by 0 or 1, correspond to making a choice of value for variable x_i, and in which the leaves are labelled by the value of the function corresponding to the input represented by the sequence of choices for values of the input variables along the path from the root to that leaf. Identifying isomorphic subtrees wherever possible, we obtain the OBDD representation of the function, a dag which is canonical for the given variable order. It can be shown that operations on boolean functions, such as pointwise conjunction ($(f_1 \wedge f_2)(\mathbf{x}) = f_1(\mathbf{x}) \wedge f_2(\mathbf{x})$ for all $\mathbf{x}$) and existential quantification (also called projection) $(\exists x_1 f)(x_2, \ldots, x_n) = f(0, x_2, \ldots, x_n) \vee f(1, x_2, \ldots, x_n)$) can be performed based at the level of the OBDD representation of the function. We refer the reader to texts such as [7] for a more detailed exposition of OBDD's and their use in model checking.

A set of states may be represented by its characteristic function, and hence by an OBDD. The OBDD for $S_i(s)$, where $s \in I$, is easily constructed, as is an OBDD representing the transition relation generated by a joint protocol $\mathbf{P}$. Similarly, given an OBDD representing $S_i(\rho)$, and the assignment representing the state s, it is possible (by the application of intersection, projection and renaming operations), to compute an OBDD representing $S_i(\rho s)$. Thus, a sequence of operations on OBDD's results in an OBDD representing $S_i(\rho)$ for the trace ρ at which we wish to verify φ.

The problem of model checking at a trace can be understood as a kind of local model checking, which verifies a formula on a particular point of a system. We may also formulate a kind of global model checking, that verifies a formula at all runs. We say that a joint protocol $\mathbf{P}$ *realizes* a specification $\varphi \in \mathcal{L}_{\{\bigcirc, U, K_1, \ldots, K_n\}}$ in an environment E if, for all runs r of $\mathcal{I}(\mathbf{P}, E)$, we have $\mathcal{I}, (r, 0) \models \varphi$.

It is a consequence of results of van der Meyden and Shilov [23] that it is decidable to determine whether a joint protocol (represented as a finite state automaton) realizes a specification $\varphi \in \mathcal{L}_{\{\bigcirc, U, K_1, \ldots, K_n\}}$ in a finite environment E. (The framework of that paper does not involve protocols, but these may be incorporated into the environment to yield the formulation of that paper.) In the case of specifications involving a single agent, ideas closely related to those just developed may be used to apply OBDD's to the implementation of the decision procedure.

We confine ourselves here to a specific class of temporal specifications, viz. specifications of the form $\bigcirc^k \varphi$, where $\varphi \in \mathcal{L}_{\{K_i\}}$. This class of specifications suffices for the example considered in the next section. One way to determine whether such a formula is realized is to check whether φ holds at all traces of length k. However, there may be exponentially many such traces, and we may often do better than this by proceeding as follows. Define X_k to be the set of sets $S_i(\rho)$, where ρ is a trace of $\mathcal{I}(\mathbf{P}, E)$ of length k. The set X_1 is readily computed, and $X_{k+1} = \{G(T, s) \mid T \in X_k \text{ and } s \in S\}$. Now, by the above, $\bigcirc^k \varphi$ is realized in $\mathcal{I}(\mathbf{P}, E)$ if for all $T \in X_k$ we have $M(T), s \models K_1 \varphi$ for an arbitrary $s \in T$. This calculation can be represented using OBDD's much along the lines described above.

In general, the size of the set X_k may grow exponentially in k. However, we may note that each sequence of observations that agent i may make in the k steps of the protocol determines an element of X_k. Let $\mathbf{V} = V_1, \ldots, V_k$ be a sequence of variables, where each V_l is a sequence of variables representing the observation that agent i makes at time l. Let $\mathbf{Z}$ be a sequence of variables representing a state of E. Then we may represent X_k as a boolean function $f_k(\mathbf{Z}, \mathbf{V})$, such that $f_k(\mathbf{Z}, \mathbf{V}) = 1$ iff the state represented by $\mathbf{Z}$ is in the element of X_k corresponding to the sequence of observations $\mathbf{V}$. This function f_k may also be represented by a OBDD, which may also be computed incrementally as follows.

Let $tr(\mathbf{Z}, \mathbf{Z}')$ be the Boolean function such that $tr(\mathbf{Z}, \mathbf{Z}') = 1$ iff there exists a transition from the state represented by $\mathbf{Z}$ to the state represented by $\mathbf{Z}'$ by a joint action $\mathbf{a}$ enabled by the protocol. Let $Ovar_i(\mathbf{Z}, V)$ be the Boolean function such that $Ovar_i(\mathbf{Z}, V) = 1$ iff the sequence of Boolean values for agent i's observable variables in the state represented by $\mathbf{Z}$ is equal to the sequence of values for the variables V. Let $Ini(\mathbf{Z})$ be the Boolean function such that $Ini(\mathbf{Z}) = 1$ iff $\mathbf{Z}$ represents an initial state.

Then, the Boolean function $f_1(\mathbf{Z}, V_1)$ representing X_1 can be defined by

$$Ini(\mathbf{Z}) \wedge Ovar_i(\mathbf{Z}, V_1);$$

and the Boolean function $f_{k+1}(\mathbf{Z}', \mathbf{V} \cdot V_{k+1})$ can be defined by

$$\exists \mathbf{Z}(f_k(\mathbf{Z}, \mathbf{V}) \wedge Ovar_i(\mathbf{Z}', V_{k+1}) \wedge tr(\mathbf{Z}, \mathbf{Z}')),$$

where $\exists \mathbf{Z}$ means the existential Boolean quantifiers for all variables in $\mathbf{Z}$.

Now, consider a formula $\bigcirc^k \varphi$ where $\varphi \in \mathcal{L}_{K_i}$. We show how to use f_k to determine if $\bigcirc^k \varphi$ is realized. As noted in Lemma 3.2, satisfaction of formulas of $\mathcal{L}_{K_i}$ at a point (r, k) depends only on the state $r(k)$ and the element $S_i(r[0 \ldots k])$ of X_k. Moreover, it can be seen that $S_i(r[0 \ldots k])$ depends only on the sequence of observations $r_i(k)$. We define a function $T_\varphi(\mathbf{Z}, \mathbf{V})$ that intuitively represents that the formula φ holds at some (equivalently, every) point (r, k) of $\mathcal{I}(\mathbf{P}, E)$ such that the state $r(k)$ is represented by $\mathbf{Z}$ and the sequence of observations $r_i(k)$ is represented by $\mathbf{V}$. The definition is by induction on the construction of φ. For formulas $K_i \psi$, the definition is

$$T_{K_i \psi}(\mathbf{Z}, \mathbf{V}) = \forall \mathbf{Z}'(f_k(\mathbf{Z}', \mathbf{V}) \Rightarrow T_\psi(\mathbf{Z}', \mathbf{V}));$$

the remaining cases are obvious. Using this function, the validity of

$$\forall \mathbf{V}(\exists \mathbf{Z}(f_k(\mathbf{Z}, \mathbf{V})) \Rightarrow \forall \mathbf{Z}(f_k(\mathbf{Z}, \mathbf{V}) \Rightarrow T_\varphi(\mathbf{Z}, \mathbf{V})))$$

expresses that $\bigcirc^k \varphi$ is realized. Note that not all sequences of observations will occur in the system: the formula $\exists \mathbf{Z}(f_k(\mathbf{Z}, \mathbf{V}))$ restricts the quantification to $\mathbf{V}$ that actually occur, i.e., for which there exists a trace ρ such that $S_i(\rho)$ is nonempty. But for all other $\mathbf{V}$ we have that $f_k(\mathbf{Z}, \mathbf{V}) = 0$ for all $\mathbf{Z}$. Hence, the above expression is equivalent to the simpler expression

$$\forall \mathbf{V}(\forall \mathbf{Z}(f_k(\mathbf{Z}, \mathbf{V}) \Rightarrow T_\varphi(\mathbf{Z}, \mathbf{V}))).$$

The computation can be done using OBDD's. For small k, the number of variables in $\mathbf{V}$ is manageable, and this approach eliminates the need to consider an exponential number of elements of X_k.

We remark that the above approach is not the only possible way to use OBDD's for this class of formulas. Another would be to directly represent the relation $\mathcal{I}(\mathbf{P}, E), \rho \models \varphi$, where ρ is a trace and $\varphi \in \mathcal{L}_{K_i}$, by a function of a sequence of variables $\mathbf{Z}_0, \ldots, \mathbf{Z}_k$ (where each $\mathbf{Z}_l$ represents a state) representing the trace ρ. The advantage of the above is that it requires fewer variables.

Indeed, further savings in the number of variables used in the OBDD may be possible. In many practical circumstances, we may find that the formula φ is independent of many of the variables in $\mathbf{V}$. For example, suppose that the state of the environment includes a boolean variable x observable to agent 1, the protocol of interest begins by assigning a random value to x, and we wish to model check the formula $\bigcirc K_1 x$. Then V contains variables x_0 and x_1 representing the value of x at time 0 and time 1. However, the value of the formula $\bigcirc K_1 x$ is clearly independent of x_0. Such independent variables can be eliminated from the arguments of the function f_k, potentially resulting in much faster calculations of the final OBDD. Our results in the next section show that the savings from this optimization can be considerable.

In the general case, for arbitrary formulas of $\mathcal{L}_{\{\bigcirc, U, K_i\}}$, we would need to construct a Büchi automaton whose states contain OBDD's representing the set of states an agent considers possible. We will not attempt to describe the details here. We remark that for formulas involving the knowledge of more than one agent, the algorithm establishing Theorem 3.1 involves not just sets of states, but hereditarily finite sets of states: how to apply OBDD's to these is presently less clear.

4. Verifying the Dining Cryptographers Protocol

We have developed a system based on the ideas above, and applied it to verify the Dining Cryptographers protocol [6], a protocol for anonymous broadcast. Chaum introduces this protocol by the following story:

> Three cryptographers are sitting down to dinner at their favorite three-star restuarant. 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 the dinner, or it might have been the NSA (US National Security Agency). The three cryptographers respect each other's right to make an anonymous payment, but they wonder if the NSA is paying.

Chaum shows that the following protocol can be used to solve the dining cryptographers' quandary. The protocol assumes that at most one cryptographer is paying.

1. Each cryptographer flips an unbiased coin behind his menu, between him and the cryptographer to his right, so that only the two of them can see the outcome.

2. Each cryptographer then states aloud whether the two coins that he can see - the one he flipped and the one his left-hand neighbour flipped – fell on the same side or different sides.

2e. As an exception to the previous step, if one of the cryptographers is the payer, he states the opposite of what he sees.

It can be shown that after running this protocols, all the cryptographers are able to determine whether it was the NSA or one of the cryptographers who paid for dinner. Specifically, an even number of differences uttered at the table indicates that NSA is paying, an odd number of differences indicates that a cryptographer is paying. Significantly, if a cryptographer is paying neither of the other two learns anything from the utterances about which cryptographer it is.

To formalize this protocol in our framework, let us first consider the specification of the problem. Introducing a proposition $\mathtt{paid}_i$ for each cryptographer $i = 1 \dots 3$, expressing that cryptographer i has paid for the dinner, we see that the following formula expresses Chaum's requirements in the case of cryptographer 1:

$$\begin{gathered} \neg \mathrm{paid}_1 \Rightarrow \\ (K_1(\neg \mathrm{paid}_1 \wedge \neg \mathrm{paid}_2 \wedge \neg \mathrm{paid}_3) \\ \vee \\ K_1(\mathrm{paid}_2 \vee \mathrm{paid}_3) \wedge \neg K_1(\mathrm{paid}_2) \wedge \neg K_1(\mathrm{paid}_3)) \end{gathered}$$

That is, if cryptographer 1 did not pay, then he knows either that no cryptographer paid, or knows that one of the other two paid, but does not know which.[2] The specifications for the other cryptographers are similar. To verify the protocol, it suffices to consider only the specification for cryptographer 1 above, because of symmetry considerations.

To capture the environment in which the cryptographers operate, we use a shared variables model, in which all variables are of boolean type and the agents have the ability to read and write shared variables, and can do so simultaneously. (The situation does not arise in the dining cryptographers, but a simultaneous write of a variable V of the environment by a number of different agents results in V having value equal to one of the values written, with the choice being nondeterministic.) The details of the modelling are straightforward, so we do not expand on this here.

The protocols run by the agents are described in a simple programming language. The representation of the dining cryptographers protocol is shown in Figure 1. The program consists of an initial declaration section followed by the body of the protocol code. The declaration section consists of three parts. The first, `environment_variables` part, lists a set of of variables which must be present in any environment in which this protocol can operate. (The environment may have more than these variables.) Some of the variables can be parameterized, with the parameter functionally depending on the agent, with the functions evaluated when the protocol is instantiated in constructing a particular arrangement of agents. The expression `I` in the protocol refers to the agent itself. Thus, in an instance of the protocol in which agent `I` has name `1`, the `environment_variables` declaration requires that the environment contain a variable `paid(1)`. Similarly, expressions such as `chan(left(I))` (which represents the channel the agent uses to communicate with the agent to the left) are bound to particular variables (e.g `chan(0)`) in creating a particular instantiation of the protocol. The expression $\{$`said(J):bool for agent(J)`$\}$ is like an array: it says that there must exist an environment variable `said(J)` for each agent J in the instance.

The `local_variables` declaration section lists variables which are accessible only to the agent. This section has a `where` section giving a constraint on the initial values of the local variables. In the dining cryptographers protocol, we have taken all local variables to be initially false. The `observable` section lists which variables are observable to the agent. The observation function O_i for agent i is derived from this list, and simply maps a state comprised of an assignment to all variables in the system to the restriction of this assignment to the set listed in the local variables declaration for agent i. Typically, all local variables will be observable, but we do not insist upon this.

Given this protocol description, a description of the shared variables environment and information to create a particular instance of the model checking problem problem (in the case of the dining cryptographers, this consists of the fact that there should be three cryptographers, and that the communication channels form a ring), our system is able to verify, using the OBDD techniques described in the previous section, the formula $\bigcirc^4 \varphi$ where φ is the specification above for cryptographer 1. (Note that the protocol takes 4 steps in our modelling.)

We have run the verification using the OBDD approach described above for several variants of the protocol, in which the number of cryptographers seated around the table varies. The runtimes are reported in Table 1. [3] OBDD size is sensitive to variable ordering. One of the capabilities of the OBDD package used is to dynamically reorder variables to try to minimize the OBDD size. Using a fixed variable ordering allowed only a small instaces to be verified,

2 We remark that the protocol is actually correct in a slightly stronger sense, viz., if one of the cryptographers paid, the others consider each of the cryptographers (except themselves) to be the payer with equal probability. Clearly our language lacks the expressiveness needed to capture this, though it would make for an interesting and useful extension to add probabilistic features.

3 Times reported are for a Pentium IV PC at 2.4 GHz with 512MB RAM, using Longs BDD package `http://www-2.cs.cmu.edu/~modelcheck/bdd.html`.

```
[ environment_variables
    paid(I): bool;
    chan(left(I)):bool;
    chan(right(I)):bool;
    {said(J):bool for agent(J)}
]
[ local_variables
    coin_left:bool;
    coin_right:bool;
  where all_false
]
[ observable
    paid(I),
    coin_left,
    coin_right,
    {said(J) for agent(J)}
]

begin
  %% decide coin toss to the right
  choose
        true -> coin_right := true
     or true -> coin_right := false
  end_choose;
  %% tell it to the person to the right
  write(coin_right,chan(right(I)));
  %% see what the coin to the left is
  read(coin_left,chan(left(I)));
  %% speak
  if (coin_left <-> coin_right)
  then
      begin
        if not(paid(I))
        then write(true,said(I))
        else write(false,said(I))
      end
  else
      begin
        if not(paid(I))
        then write(false,said(I))
        else write(true,said(I))
      end
end
```

Figure 1. Protocol for a dining cryptographer

No. of Cryptographers:	3	4	5	10	20
fixed ordering	0.3	2.3	26.8	-	-
with sifting	.7	1.8	6.9	66	519

Table 1. Runtimes of Dining Cryptographers Verification (Seconds)

and the runtimes appear to grow exponentially (the case of 6 agents ran more than 10 hours.) Using the sifting reordering strategy enables larger instances to be verified, but the rate of growth remains exponential.

We have also compared the OBDD approach described above with several other approaches to the verification. Explicit construction of the Kripke structure comprised of traces of length 5 in the case of 3 cryptographers takes over 15 minutes. (Note that the `chan` and `said` variables are initially unconstrained in our modelling. We could constrain them to a fixed value without change to the conclusions of the protocol, but have not done so to illustrate that it is one of the benefits of the OBDD approach that it catches, to a certain extent, such optimizations automatically. Moreover, note that because of the coin-flips, the number of traces of length 5 grows exponentially with the number of cryptographers, so we would not expect an explicit enumeration approach to be asymptotically efficient even with such optimizations.)

We have also considered a translation of the model checking problem to satisfiability of a formula in WS1S, the weak monadic second order logic of one successor, and the application of the MONA system [17] to decide satisfiability of this formula. On this approach, the formula, even in the case of 3 cryptographers, proves to have quantifier nesting too deep for MONA to run successfully (it is unable to complete the Buchi automaton construction). The problem can also be translated to a QBF validity problem, though at the cost of a much larger number of quantified variables. The QBF formula also proves to be too large for MONA to handle.

As we remarked at the end of the previous section, the elimination of variables that can be identified to be independent of the formula being model checked potentially results in improvements in the efficiency of model checking. In the case of the dining cryptographers protocol, the improvements turn out to be considerable. Notice that the function f_k has as an argument a variable x_n for each of the observable variables x, and each time $n = 0 \dots 4$. However, from an analysis of the protocol (by hand) we can see that

1. the variable `paid(I)` is constant, so needs only to be included once.

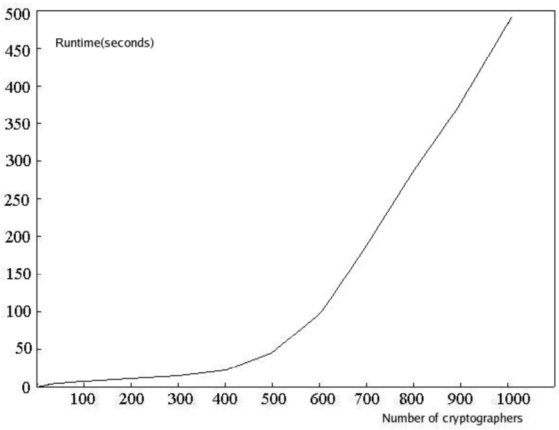

Figure 2. Runtime vs cryptographers with a reduced variable set

2. only the final observed value of the variables `said(J)` need be included: the earlier values are either random and independent of the final values of `paid(J)`, or are identical to the final value of `said(J)`. Moreover, the time at which the transition to the final value is made is common knowledge, so no agent learns anything from observing this time.
3. A similar remark to the previous applies to the values of `coin_left` and `coin_left`

When we reduce the set of variables accordingly, we obtain an OBDD calculation which gives dramatically improved runtimes, depicted in Figure 2, which plots runtimes in seconds (vertical axis) versus number of cryptographers. Whereas the runtime results above appear to be exponential, these results are piecewise linear, with the transition corresponding to the point where the computation can no longer run in main memory and starts swapping to disk.

We do not yet have a tool that can identify such optimization opportunities automatically, but it is clear from these results that the incorporation of static analysis techniques to eliminate redundancies prior to model checking is likely to have a significant impact on the performace of model checkers for the logic of knowledge.

5. Related work

Perhaps closest to our work in this paper are some applications of model checking in the artificial intelligence literature. Gorogiannis and Ryan [12] has explored the use of binary decision diagrams for the implementation of belief revision operators. While the modality of concern here is belief rather than knowledge, and there is no notion of protocol, the revision operators considered bear some resemblance to our operator G. A number of authors have also considered model checking for logics involving temporal and epistemic modalities [25, 3, 18]. The main difference between these works and ours is that they use semantics for knowledge that are weaker than the perfect recall definition we have studied. Very roughly, these works can be characterised as dealing with notions of knowledge that define an agent's knowledge from its current observation, rather than its history of observations, as we have done. While this allows the application of standard model checking systems, it is unsuited to security applications and applications where one wishes to optimize agents' use of information.

In most cases, the focus of the existing literature [20, 24, 8] on model checking security protocols has been on protocols employing cryptography to obtain secure channels. Information flow is important in such protocols. However, the approach taken in these works does not treat information as a first class citizen, but attempts to reduce it to state properties so that temporal verification technology can be applied. This is done by capturing a number of specific ways in which an agent may come to know particular types of information, following an approach of Dolev and Yao [9]. An agent's knowledge is modelled by means of a list of message components possessed by the agent. Information acquisition is captured by rules relating these components. For example, an agent in possession of a private key K, that has received a message $\{M\}_{K^{-1}}$, constructed by encrypting the message M with the corresponding public key K^{-1}, is taken also to be in possession of the message M. This can be modelled, e.g., by allocating a bit b_M in the state representation to encode this fact. The actions of the agent are then given preconditions that refer to such bits. E.g., a precondition of the agent constructing a message containing M as a component is that b_M be true.

This modelling captures some aspects of the way that information flows in the execution of security protocols, and can be used to detect the existence of certain types of attacks using *temporal* model checkers. An error trace produced by a model checker based on this approach shows that it is possible for an adversary to deceive the principals of the protocol into entering protocol states held by the protocol designer to indicate successful establishment of a security goal, when this security goal has not in fact been met. This is proof that the protocol is insecure.

However, when model checking using this approach fails to report an error — even when a complete search of the traces of the protocol has been performed — there is no guarantee that the protocol is correct. As can be seen from the dining cryptographers protocol, information flows in concurrent systems in remarkably subtle ways, and the agent may have other ways of extracting information from its observations than those that have been explicitly modelled.

It is possible to do a partial verification of the dining cryptographers protocol using temporal model checking tools, by checking that the parity of the assertions made by the dining cryptographers captures whether the NSA or one of the cryptographers has paid. However, this requires that the concrete test for the knowledge condition be explicitly supplied. In general, this is undesireable in a model checker for knowledge, since we will often want to use the tool to answer the question "is φ known in circumstances X" without having a particular test for knowledge in mind. We may also be wrong about the test corresponding to a particular condition for knowledge, and want the tool to tell us so if it is not necessary or not sufficient. Moreover, this approach cannot handle the remainder of the dining cryptographers specification: that in the event the payer is one of the cryptographers, the others *do not know* which cryptographer it is.

A number of works have previously studied the formal verification of anonymity protocols, but the motivations and techniques of these works are rather different from ours. Schneider and Sidiropolous [26] use process algebraic tools (CSP). One of their main objectives is to study the notion of *anonymity.* They give a definition of this specific notion as an equation relating terms of operations on the sets of traces of process algebraic terms, and then use the FDR model checker to verify the anonymity of the payer in the dining cryptographers protocol. Unlike the Dolev-Yao based approaches discussed above, their approach *is* information theoretic. However, whereas we have presented a technique for verifying a broad class of knowledge properties of protocols, and an explicit logical notation for expressing these properties, they focus on a quite specific class of properties. In order to verify that the protocol achieves the goal that the diners know whether the NSA or one of the cryptographers has paid, they follow the technique of explicitly supplying the parity test for this condition, whose disadvantages we have discussed above. (Their representation of this part of the specification is implicit, and although they discuss the case of four cryptographers, they appear to have performed this verification only in the case of 3 cryptographers. It is unclear how their technique scales.)

Closer to our approach to the dining cryptographers is the work of Syverson and Stubblebine [28], who use a variant of the logic of knowledge with some features resembling BAN logic [5] to specify The Anonymizer, a web proxy service that protects client identity. However, they do not present any formal analysis of the specification. It appears they have a proof theoretic method in mind, though they express an interest in model checking techniques. Halpern and O'Neill [15] have also considered anonymity protocols from the perspective of the logic of knowledge.

Another interesting work on verification of an anononymity protocol is Shmatikov's analysis of the Crowds protocol [27] using probabilistic model checking. This work is also information theoretic, using conditionalization to capture states of information. Probability plays an important role in security protocols, and we intend to add this capability in future work. One could also study the dining cryptographers from the probabilistic perspective. We note that use of probability theory does not eliminate the need for the logic of knowledge: settings with nondeterministic events as well as probabilistic events require both the logic of knowledge and probability theory [16]. (E.g., it is reasonable to view the determination of who has paid in the initial state of the dining cryptographers as nondeterministic, and not involving probability.) Our work in this paper is a first step towards handling the combination of these dimensions.

We intend in future work to consider other protocols (e.g protocols for secret sharing and game theoretic examples), to which our model checker can be directly applied. A number of obstacles need to be overcome before we can apply the ideas of the present paper more generally to to verification of knowledge properties in security protocols. One is that such protocols usually assume a large key space: we would need to apply data independence techniques [29, 19] to reduce such key spaces to a tractably finite set. How to do so for specifications in the logic of knowledge remains to be worked out. We also expect that any automated approach would have to make use of the *perfect cryptography* assumption. (Not only does the subtle interaction of complexity theoretic and probabilistic notions underlying modern cryptography extend beyond the expressive power of current model checking technology, but the appropriate semantics for knowledge in the context of protocols making use of these notions remains to be worked out.) Useful in this regard may be the semantics for BAN logic [5] developed by Abadi and Tuttle [1], which already draws on the semantics of knowledge used in the literature on reasoning about knowledge. The BAN logic rules have already been combined with a model checker for the logic of belief and time by Benerecetti and Giunchiglia [2]. However, this work can perhaps be best characterized as transforming a validity check for a BAN logic formula into a branching time logic model checking problem. Their approach lacks a number of the key features that make our approach attractive: the explicit modelling of observations and the role of perfect recall. We note that BAN logic itself is unsuited to the analysis of protocols like the dining cryptographers protocol because it is tailored for authentication protocols only, and makes no attempt at a complete modelling of information flow. Moreover no completeness proof for BAN logic has ever been attempted.

Our longer term aim in this work is the development of a tool for automated analysis of knowledge in concurrent systems. Such a tool would have applications going beyond the computer aided verification of security protocols: we are also interested in providing automated support for the analysis of knowledge in distributed systems protocols and game theoretic examples, and the verification and compilation of knowledge-based programs [10].

References

[1] M. Abadi and M. Tuttle. A semantics for a logic of authentication. In *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing*, pages 201–216, 1991.

[2] M. Benerecetti and F. Giunchiglia. Model checking security protocols using a logic of belief. In S. Graf and M. Schwartzbach, editors, *Proc. 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2000)*, volume LNCS No. 1785, pages 519–534, Berlin, 2000. Springer.

[3] M. Benerecetti, F. Giunchiglia, and L. Serafini. A model checking algorithm for multi-agent systems. In J. Muller, M. Singh, and A. Rao, editors, *Intelligent Agents V*, volume LNAI Vol. 1555. Springer-Verlag, Berlin, 1999.

[4] J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proc. Symposium on Logic in Computer Science*, pages 428–439. IEEE, 1990.

[5] M. Burrows, M. Abadi, and R. M. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1), 1990.

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

[7] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2000.

[8] E. Clarke, S. Jha, and W. Marrero. A machine checkable logic of knowledge for specifying security properties of electronic commerce protocols. In *LICS Security Workshop*, 1998.

[9] D. Dolev and A. Yao. On the security of public-key protocols. *Communications of the ACM*, 29(8):198–208, August 1983.

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

[11] S. Goel, M. Robson, M. Polte, and E. G. Sirer. Herbivore: A scalable and efficient protocol for anonymous communication. Technical Report TR2003-1890, Cornell University Computing and Information Science, Feb 2003.

[12] N. Gorogiannis and M. D. Ryan. Implementation of belief change operators using bdds. *Studia Logica*, 70(1), 2001.

[13] J. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. *J. ACM*, 37(3):549–587, 1990.

[14] J. Halpern and M. Y. Vardi. Model checking vs. theorem proving: A manifesto. Technical Report RJ 7963, IBM Almaden Research Center, 1991. An extended version of a paper in *Proc. 2nd Int. Conf. on Principles of Knowledge Representation and Reasoning, 1991*.

[15] J. Y. Halpern and K. R. O'Neill. Anonymity and information hiding in multiagent systems. In *IEEE Computer Security Foundations Workshop*, pages 75–88, 2003.

[16] J. Y. Halpern and M. R. Tuttle. Knowledge, probability, and adversaries. *J. ACM*, 40(4):917–962, 1993. A preliminary version appears in *Proceedings of the 8th ACM Symposium on Principles of Distributed Computing*, 1989, pp. 103–118.

[17] J. G. Henriksen, J. L. Jensen, M. E. Jorgensen, N. Klarlund, R. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic second-order logic in practice. In *Tools and Algorithms for Construction and Analysis of Systems, First INternational Workshop, TACAS'95*, LNCS Vol. 1019, pages 89–110, 1995.

[18] W. van der Hoek and M. Wooldridge. Model checking knowledge and time. In *9th Workshop on SPIN (Model Checking Software)*, Grenoble, April 2002.

[19] M. L. Hui and G. Lowe. Simplifying transformations for security protocols. In *Proc. 12th IEEE Computer Security Foundations Workshop*, pages 32–43, 1999.

[20] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In Margaria and Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, Vol 1055 of Lecture Notes in Computer Science, pages 147–166. Springer Verlag, 1996.

[21] R. van der Meyden. Knowledge-based programs: On the complexity of perfect recall in finite environments. In *Proc. Conf. on Theoretical Aspects of Rationality and Knowledge*, pages 31–50, 1996.

[22] R. van der Meyden. Common knowledge and update in finite environments. *Information and Computation*, 140(2):115–157, 1998. A preliminary version of this paper appears in *Proc. Conf on Theoretical Aspects of Reasoning about Knowledge*, 1994.

[23] R. van der Meyden and N. Shilov. Model checking knowledge and time in systems with perfect recall. In *Proc. Conf. on Software Technology and Theoretical Computer Science*, Springer LNCS No 1738, pages 262–273, Berlin, 1999.

[24] J. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murphi. In *IEEE Symp. on Security and Privacy*, pages 141–153, 1997.

[25] A. Rao and M. Georgeff. A model theoretic approach to the verification of situated reasoning systems. In *Proc. 13th International Joint Conference on Artificial Intelligence*, pages 318–324, 1993.

[26] S. Schneider and A. Sidiropolous. Csp and anonymity. In E. B. et al, editor, *Computer Security - ESORICS 96, 4th European Symposium on Research in Computer Security, Rome, Italy, September 25-27, 1996, Proceedings*, volume 1146 of *Lecture Notes in Computer Science*, pages 198–218. Springer, 1996.

[27] V. Shmatikov. Probabilistic analysis of anonymity. In *Proc. 15th IEEE Computer Security Foundations Workshop*, 2002.

[28] P. Syverson and S. Stubblebine. Group principals and the formalization of anonymity. In *FM'99 Formal Methods*, volume No. 1708, Vol I of *Springer LNCS*, pages 314–333, 1999.

[29] P. Wolper. Expressing interesting properties of programs in propositional temporal logic. In *Proc. 13th ACM Symp. on Principles of Programming Languages*, pages 184–192, St. Petersburgh, January 1986.