Symbolic Model Checking of Probabilistic Knowledge*

[Extended Abstract]

Xiaowei Huang, Cheng Luo, and Ron van der Meyden
Computer Science and Engineering
University of New South Wales, Australia
{xiaoweih,luoc,meyden}@cse.unsw.edu.au

ABSTRACT

This paper describes an algorithm for model checking a fragment of the logic of knowledge and probability in multi-agent systems, with respect to a perfect recall interpretation of knowledge and agents' subjective probability. The algorithm has been implemented in the epistemic model checker MCK. Some experiments with the implemented algorithm are reported, in which some properties of agents' probabilistic knowledge are verified in two security protocols: Chaum's Dining Cryptographers protocol, and a protocol for Oblivious Transfer due to Rivest.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification; F.4.1 [**Mathematical Logic**]: Modal Logic; G.3 [**Mathematics of Computing**]: Probability and Statistics

General Terms

Theory

Keywords

Model Checking, Logic of Knowledge, Probability

1. INTRODUCTION

Model checking [7] is an approach to the verification of systems designs, in which one describes the system to be verified as a model of some logic, expresses the property to verified as a formula in that logic, and then checks using automated algorithms that the formula holds in the model. In general, this verification problem has high computational complexity, but a range of heuristic algorithms have been developed that render the approach feasible in practice for problems of industrial relevance. When feasible, model checking has advantages over other approaches to verification: compared to testing – which gives only a partial guarantee of correctness – it yields a complete guarantee of correctness (or a guarantee that a counter-example to correctness will be found); compared to theorem proving approaches – which are laborious and require highly trained staff – it is fully automated.

Model checking has been most extensively studied for the case where the system can be represented as a finite state automaton, and the specification expressed in a linear or branching time temporal logic; this type of model checking is now a standard verification methodology in computer hardware development. Model checking has also been studied for temporal logics in which the temporal operators are extended with probabilistic expressiveness, in the form of a type of objective probability: the models considered are either discrete-time Markov chains, continuous-time Markov chains or Markov Decision processes [16]. More recently, model checkers have emerged for which the base logic is a logic of knowledge and time [12, 19, 17, 26] or dynamic update logic [9].

To date, there has been no implemented model checking system that deals with the logic of knowledge, time and the *subjective* probability, i.e., probability relative to agent knowledge. The model checking system that we develop for this combination in this paper is based on the logic of knowledge and probability [11], which adds to the logic of knowledge the ability to express facts about probability (implicitly) conditioned on what an agent knows, using expressions such as "Agent i's probability of formula ϕ is greater than $1/2$". We follow one of the semantics of Halpern and Tuttle [15], in which agent's probabilistic knowledge is determined using the assumption of synchronous perfect recall. The models that we handle are in the form of discrete-time Markov chains. Our approach to model checking this logic is a generalisation of an approach, due to van der Meyden and Su [28], using symbolic model checking techniques for model checking a fragment of the logic of knowledge and time with perfect recall.

The structure of the paper is as follows. We introduce the semantics for the logic of knowledge, probability and time in Section 2. In Section 3, we describe a fragment of this logic, and our approach to model checking this fragment. Section 4 briefly describes the implementation of the algorithm in the model checker MCK. Section 5 then gives a number of applications of the implemented system to verification of security protocols. We discuss related work in Section 6 and make some concluding remarks in Section 7.

2. PROBABILISTIC KNOWLEDGE

We describe in this section the semantic setting for the model checking problem we consider. We largely follow the definitions of probabilistic interpreted systems [13], in which we model a set of agents $Agt = \{1, \ldots, n\}$ operating in an environment e. At all times, each agent is assumed to be in some *local* state, which records all the information that the agent can access at that time. The environ-

*Work supported by Australian Research Council Linkage Grant LP0882961 and Defence Research and Development Canada (Valcartier) contract W7701-082453

TARK 2011, July 12-14, 2011, Groningen, The Netherlands.
 ISBN 978-1-4503-0707-9, $10.00.

ment e records "everything else that is relevant". Let S be the set of environment states and let L_i be the set of local states of agent $i \in Agt$. A *global* state of a multi-agent system is a $(n+1)$-tuple $s = (s_e, s_1, \ldots, s_n)$ such that $s_e \in S$ and $s_i \in L_i$ for all $i \in Agt$.

Time is represented discretely using the natural numbers $\mathbb{N}$. A *run* is a function $r : \mathbb{N} \to S \times L_1 \times \ldots \times L_n$ from time to global states. A pair (r, m) consisting of a run r and time m is called a *point*. If $r(m) = (s_e, s_1, \ldots, s_n)$ then we define $r_e(m) = s_e$ and $r_i(m) = s_i$ for $i \in Agt$. If r is a run and m a time, we write $r_e[0..m]$ for $r_e(0) \ldots r_e(m)$. A *system* is a set $\mathcal{R}$ of runs. We call $\mathcal{R} \times \mathbb{N}$ the *set of points* of the system $\mathcal{R}$. Relative to a system $\mathcal{R}$, we define the set $\mathcal{K}_i(r, m) = \{(r', m') \in \mathcal{R} \times \mathbb{N} \mid r'_i(m') = r_i(m)\}$ to be the set of points that are, for agent i, indistinguishable from the point (r, m), and $\mathcal{R}(U) = \{r \in \mathcal{R} \mid \exists m : (r, m) \in U\}$ to be the set of runs in $\mathcal{R}$ going through some point in the set $U \subseteq \mathcal{R} \times \mathbb{N}$. Agents have *synchronous perfect recall* in system $\mathcal{R}$ if there exists a set O (of *observations*) such that for each point (r, m) of $\mathcal{R}$ and agent i, the local state $r_i(m)$ is a sequence of exactly $(m+1)$ elements of O, and $r_i(m+1) = r_i(m) \cdot o$ for some $o \in O$.

A *probability space* is a triple (W, F, μ) such that W is a set, called the *carrier*, $F \subseteq \mathcal{P}(W)$ is a set of *measurable* sets in W, closed under union and complementation, and $\mu : F \to [0, 1]$ is a *probability measure*, such that $\mu(W) = 1$ and $\mu(U \cup V) = \mu(U) + \mu(V)$ if $U \cap V = \emptyset$. As usual, we define the conditional probability $\mu(U|V) = \mu(U \cap V)/\mu(V)$ when $\mu(V) \neq 0$.

A *probabilistic interpreted system* is a tuple $(\mathcal{R}, PR_1, \ldots, PR_n, \pi)$ such that $\mathcal{R}$ is a system, each PR_i is a function mapping each point (r, m) of $\mathcal{R}$ to a probability space $PR_i(r, m)$ in which the carrier is a set of points of $\mathcal{R}$, and $\pi : \mathcal{R} \times \mathbb{N} \to \mathcal{P}(Prop)$ is an interpretation of some set $Prop$ of atomic propositions.

We will work with systems in which agents have a common prior on the set of runs and synchronous perfect recall. Let $\mathcal{R}$ be a system, let $\mathbf{P} = (\mathcal{R}, F_{\mathcal{R}}, \mu_{\mathcal{R}})$ be a probability space on the system $\mathcal{R}$, and let π be an interpretation on $\mathcal{R}$. We assume that for each point (r, m) of $\mathcal{R}$ and agent i, the set $\mathcal{R}(\mathcal{K}_i(r, m))$ is measurable and $\mu_{\mathcal{R}}(\mathcal{R}(\mathcal{K}_i(r, m))) > 0$. we then derive a probabilistic interpreted system $\mathcal{I}(\mathcal{R}, \mathbf{P}, \pi) = (\mathcal{R}, PR_1, \ldots, PR_n, \pi)$ such that PR_i associates with each point (r, m) the probability space $PR_i(r, m) = (\mathcal{K}_i(r, m), F_{i,r,m}, \mu_{r,m,i})$ such that $F_{i,r,m}$ is the set of $U \in \mathcal{P}(\mathcal{K}_i(r, m))$ such that $\mathcal{R}(U) \in F_{\mathcal{R}}$, and

$$\mu_{r,m,i}(U) = \mu_{\mathcal{R}}(\mathcal{R}(U) \mid \mathcal{R}(\mathcal{K}_i(r, m))) .$$

It is easily shown that it follows from the assumptions that $PR_i(r, m)$ is a probability space. Intuitively, at each point, each agent has a probability space in which the carrier is the set of points $\mathcal{K}_i(r, m)$ that it considers possible, and which is related to the prior on runs by a type of temporal conditioning. (See [13] for a detailed explanation of these definitions and this point.)

To specify properties of probabilistic interpreted systems, we work with a logic that combines temporal operators, knowledge operators, and probabilistic operators. Its syntax is given by the grammar

$$\phi \ ::= \ p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid X\phi \mid K_i\phi \mid L(P_1, \ldots, P_k) \bowtie c$$

where $p \in Prop$, c is a rational constant, $\bowtie$ is a relation symbol in the set $\{\leq, <, =, >, \geq\}$, and $L(P_1, \ldots, P_k)$ is a rational linear combination of the P_j, each of which is of one of the forms $\mathtt{Pr}_i\phi$, $\mathtt{Pr}_i(\phi_1|\phi_2)$, $\mathtt{Prior}_i\phi$ or $\mathtt{Prior}_i(\phi_1|\phi_2)$. Intuitively, formula $K_i\phi$ expresses that agent i knows ϕ, $X\phi$ expresses that ϕ holds at the next moment of time, the expression $\mathtt{Pr}_i\phi$ represents agent i's current probability of ϕ, the expression $\mathtt{Pr}_i(\phi_1|\phi_2)$ represents agent i's current probability of ϕ_1 under the condition ϕ_2, the expression $\mathtt{Prior}_i\phi$ represents agent i's *prior* probability of ϕ, $\mathtt{Prior}_i(\phi_1|\phi_2)$ represents agent i's *prior* probability of ϕ_1 under the condition ϕ_2, and $L(P_1, \ldots, P_k) \bowtie c$ expresses that this linear combination of current and prior probabilities stands in the relation $\bowtie$ to c. The semantics of the language in a system $\mathcal{I} = \mathcal{I}(\mathcal{R}, \mathbf{P}, \pi)$ is given by interpreting formulas ϕ at points (r, m) of $\mathcal{I}$, using a satisfaction relation $\mathcal{I}, (r, m) \models \phi$. We first interpret the probability expressions at points of $\mathcal{I}$ as in Table 1.

The satisfaction relation is then defined recursively as follows, omitting the obvious boolean cases:

1. $\mathcal{I}, (r, m) \models p$ if $p \in \pi(r, m)$
2. $\mathcal{I}, (r, m) \models X\phi$ if $\mathcal{I}, (r, m+1) \models \phi$
3. $\mathcal{I}, (r, m) \models K_i\phi$ if $\mathcal{I}, (r', m') \models \phi$ for all $(r', m') \in \mathcal{K}_i(r, m)$.
4. $\mathcal{I}, (r, m) \models L(P_1, ..., P_k) \bowtie c$ if $L(c_1, ..., c_k) \bowtie c$, where $[P_j]_{\mathcal{I},(r,m)} = c_j$ for $j = 1 \ldots k$.

We already noted above that some assumptions on $\mathbf{P}$ are required for the construction of $\mathcal{I} = \mathcal{I}(\mathcal{R}, \mathbf{P}, \pi)$. The above semantics additionally assumes that the sets whose probability is evaluated in Table 1 are measurable. We introduce a more concrete setting in what follows that ensures these assumptions are met.

Although they provide a coherent semantic framework, probabilistic interpreted systems are infinite structures, and therefore not suitable as input for a model checking algorithm. We therefore work with a type of finite model called an *interpreted partially observed discrete-time Markov chain*, or PO-DTMC for short. A finite PO-DTMC for n agents is a tuple $M = (S, PI, PT, O_1, ..., O_n, \pi)$, where S is a finite set of states, $PI : S \to [0..1]$ is an initial probability function such that $\sum_{s \in S} PI(s) = 1$, component $PT : S \times S \to [0, 1]$ is a probability transition matrix, such that $\sum_{s' \in S} PT(s, s') = 1$ for all $s \in S$, and for each agent $i \in Agt$, we have an observation function $O_i : S \to O$ for agent i. Finally, $\pi : S \to \mathcal{P}(Prop)$ is an interpretation of the atomic propositions $Prop$ at the states. We treat the set of states S as the states of the environment rather than as the set of global states. Intuitively, $O_i(s)$ is the observation that agent i makes when the system is in state s; agents' local states will be derived from these observations.

We write $k_i(s) = \{s' \in S \mid O_i(s') = O_i(s)\}$ for the set of states that are observationally indistinguishable to agent i from state s. We write $s \xrightarrow{p} s'$ if $PT(s, s') = p > 0$, and $s \to s'$ if $PT(s, s') > 0$. A *path* from a state s is a finite or infinite sequence of states $\rho = s_0 s_1 \ldots$ such that $s_0 = s$ and $s_k \to s_{k+1}$ for all k less than the length of ρ. Given a path ρ, we use $\rho(m)$ to denote its $(m+1)$-th state. A *fullpath* from a state s is an infinite path from s. A path ρ is *initialized* if $PI(\rho(0)) > 0$.

We may construct several different probabilistic interpreted systems from each PO-DTMC, depending on what agents' remember of their observations. In this paper, we concentrate on a definition that assumes agents have synchronous perfect recall, denoted `spr`. Recall that runs map time to global states consisting of a state of the environment and a local state for each agent, and we interpret the states of the PO-DTMC as states of the environment. Thus, to construct a run, we use the observations to construct the local states. Given an initialized fullpath ρ, we obtain a run $\rho^{\mathtt{spr}}$ by defining the components at each time m as follows. The environment state at time m is $\rho_e^{\mathtt{spr}}(m) = \rho(m)$, and the local state of agent i at time m is $\rho_i^{\mathtt{spr}}(m) = O_i(\rho(0)) \ldots O_i(\rho(m))$, representing that the agent remembers all its observations. Note that $\rho_e^{\mathtt{spr}}[0..m] = \rho(0) \ldots \rho(m)$.

We now define a probability space on the set of runs we generate as above from the environment, using a well-known construction. Let $\mathcal{R}$ be the set of runs in M. Given a finite initialized path ρ of length $m+1$, write $\mathcal{R}(\rho) = \{r \in \mathcal{R} \mid r_e[0 \ldots m] = \rho\}$ for the set of

$$
\begin{array}{rcl}
[\mathtt{Pr}_i\phi]_{\mathcal{I},(r,m)} & = & \mu_{r,m,i}(\{(r',m') \in \mathcal{K}_i(r,m) \mid \mathcal{I},(r',m') \models \phi\}) \\
[\mathtt{Prior}_i\phi]_{\mathcal{I},(r,m)} & = & \mu_{r,0,i}(\{(r',0) \in \mathcal{K}_i(r,0) \mid \mathcal{I},(r',0) \models \phi\}) \\
[\mathtt{Pr}_i(\phi_1|\phi_2)]_{\mathcal{I},(r,m)} & = & [\mathtt{Pr}_i(\phi_1 \wedge \phi_2)]_{\mathcal{I},(r,m)} \div [\mathtt{Pr}_i\phi_2]_{\mathcal{I},(r,m)}, \text{ if } [\mathtt{Pr}_i\phi_2]_{\mathcal{I},(r,m)} \neq 0 \\
[\mathtt{Prior}_i(\phi_1|\phi_2)]_{\mathcal{I},(r,m)} & = & [\mathtt{Prior}_i(\phi_1 \wedge \phi_2)]_{\mathcal{I},(r,m)} \div [\mathtt{Prior}_i\phi_2]_{\mathcal{I},(r,m)}, \text{ if } [\mathtt{Prior}_i\phi_2]_{\mathcal{I},(r,m)} \neq 0
\end{array}
$$

Table 1: Interpreted system semantics of probabilistic operators

runs with prefix ρ for the environment part. (One may view this as a cone of runs over prefix ρ.) Let $F_{\mathcal{R}}$ be the minimal algebra with basis the sets $\mathcal{R}(\rho)$ for ρ a finite initialized path of M, i.e., the set of all sets of runs that can be constructed from the basis using union and complementation. Suppose we define the measure $\mu_{\mathcal{R}}$ on the basis sets by

$$\mu_{\mathcal{R}}(\mathcal{R}(\rho)) = PI(\rho(0)) \times \prod_{i=0}^{m-1} p_i$$

when $\rho(i) \xrightarrow{p_i} \rho(i+1)$ for $0 \leq i \leq m-1$. There is a unique extension of $\mu_{\mathcal{R}}$ to $F_{\mathcal{R}}$ that satisfies the constraints on probability measures (i.e., finite additivity and universality), and we also denote this by $\mu_{\mathcal{R}}$. It can then be shown that $\mathbf{P}(M) = (\mathcal{R}, F_{\mathcal{R}}, \mu_{\mathcal{R}})$ is a probability space on $\mathcal{R}$.

The system M gives us an interpretation π on its states, and we may lift this to an interpretation on the points (r,m) of $\mathcal{R}$ by defining $\pi(r,m) = \pi(r_e(m))$. Using the construction above, we then obtain the probabilistic interpreted system $\mathcal{I}(M) = \mathcal{I}(\mathcal{R}, \mathbf{P}(M), \pi)$. We noted above that the construction of a system $\mathcal{I}(\mathcal{R}, \mathbf{P}, \pi)$ requires an assumption on $\mathbf{P}$, and moreover, the measurability of sets arising during the evaluation of a formula introduces a further assumption. For the systems $\mathcal{I}(\mathcal{R}, \mathbf{P}(M), \pi)$ and the language we consider in this paper, these assumptions are met, as is shown in the following result.

PROPOSITION 1. *Let M be a finite PO-DTMC. Then the sets $\mathcal{R}(\mathcal{K}_i(r,m))$ are measurable and have non-zero measure. Also, for all formulas ϕ, the sets $\{(r',m') \in \mathcal{K}_i(r,m) \mid \mathcal{I}(M),(r',m') \models \phi\}$ are measurable in $PR_i(r,m)$.*

We will be interested in the problem of model checking formulas in this system. A formula ϕ is said to hold in M, written $M \models \phi$, if $\mathcal{I}(M),(r,0) \models \phi$ for all $r \in \mathcal{R}$. The model checking problem is then to determine, given a PO-DTMC M and a formula ϕ, whether $M \models \phi$.

3. A SYMBOLIC MODEL CHECKING ALGORITHM

The question of model checking the logic of knowledge and time (without probabilities) with respect to perfect recall has been studied in a number of works [27, 20], which identify various fragments for which the problem is decidable. Decidability does not imply practical feasibility, however, and for this further restrictions and special algorithms are required. An algorithm that has proved useful in practice was introduced by van der Meyden and Su [28]; it works on formulas of the form $X^k\phi$ where ϕ is a formula in which the only modal operator is K_i, for a fixed agent i. While restricted, this algorithm has been successfully applied to interesting case studies including security protocols [28, 1], card games [29] and cache coherency [2]. A typical application is to verify formulas of the form $X^k(l_i \Leftrightarrow K_i\phi)$, where ϕ is a global property and l_i is a predicate local to agent i; this class of formulas is useful for verifying implementations of knowledge-based programs [10].

In this section, we generalize this algorithm to the language of the present paper, in which we add probabilistic expressiveness. The generalization works for formulas of the form of the form $X^d\varphi$ with the only modal operators in φ being the knowledge operator K_i and the probability operators Pr_i and $\mathtt{Prior}_i$, for a single, fixed agent i. An example of such a formula is

$$X^d(K_i\varphi_1 \Rightarrow \mathtt{Pr}_i\varphi_2 \leq c)$$

expressing that after d steps, if agent i knows the fact φ_1 then it knows φ_2 with a probability no more than c. These properties are useful in characterizing the posterior probability on agent i's knowledge after some number of steps.

Note that the number of possible paths of length d in a system with states S is potentially as large as $|S|^d$, so that specifications of the form we consider refer to a structure that grows exponentially with d in the worst case; indeed, the number of states S already grows exponentially in the number of boolean variables used to define a state. A naive implementation of model checking would therefore need to work with exponentially large structures, and would not be practical. The idea underlying our algorithm is to work with *symbolic representations* of certain functions that encode the information required for model checking. Because they condense symmetries, these symbolic representations are, in practice, often more compact than the information they represent (although there is not a guarantee of this in general – the technique is merely heuristic). We first define the functions we represent, and later discuss the symbolic representation (multi-terminal OBDD's) that the algorithm uses to encode these functions.

Let i be the agent whose modal operators occur in the formula we wish to evaluate. For each number $k \geq 0$ and φ a formula in which the only modal operators are knowledge and probability operators of agent i, we define several functions $\mathtt{T}^k, \mathtt{T}^k_\varphi, \mathtt{P}^k, \mathtt{P}^k_\varphi$: each takes in as input $k+1$ observations $o_0...o_k$ of agent i, and a state $s \in S$. The definitions refer to $R_k(o_0...o_k, s)$, which is defined to be the set of runs of $\mathcal{I}(M)$ such that $r_i(k) = o_0...o_k$ and $r_e(k) = s$. The functions $\mathtt{T}^k, \mathtt{T}^k_\varphi$ return a value in $\{0,1\}$, which we may interpret as either a boolean value or a real number, and the functions $\mathtt{P}^k, \mathtt{P}^k_\varphi$ return a value in the real interval $[0,1]$.

DEFINITION 1.

1. *Let $\mathtt{T}^k(o_0...o_k, s) = 1$ if $R_k(o_0...o_k, s)$ is nonempty, otherwise $\mathtt{T}^k(o_0...o_k, s) = 0$.*
2. *Let $\mathtt{T}^k_\varphi(o_0...o_k, s) = 1$ if there exists $r \in R_k(o_0...o_k, s)$ such that $\mathcal{I}(M),(r,k) \models \varphi$, otherwise $\mathtt{T}^k_\varphi(o_0...o_k, s) = 0$.*
3. *Let $\mathtt{P}^k(o_0...o_k, s) = \mu_{\mathcal{R}}(R_k(o_0...o_k, s))$.*
4. *Let $\mathtt{P}^k_\varphi(o_0...o_k, s) = \mu_{\mathcal{R}}(\{r \in R_k(o_0...o_k, s) \mid \mathcal{I}(M),(r,k) \models \varphi\})$.*

The following theorem shows that we may characterize the model checking problem using the functions $\mathtt{T}^k$ and $\mathtt{T}^k_\varphi$.

THEOREM 1. *Let M be a PO-DTMC with states S and set of observations O for the unique agent whose modalities occur in the atemporal formula φ. Then $M \models^{\mathtt{spr}} X^d\varphi$ is equivalent to the truth of $\forall s \in S, o_0, ..., o_d \in O^{d+1} : (\mathtt{T}^d(o_0...o_d, s) \Rightarrow \mathtt{T}^d_\varphi(o_0...o_d, s))$.*

One of the keys to our algorithm is then the fact that the functions in Definition 1 satisfy some equations that enable them to be constructed by means of a mutual recursion, that can be encoded using a symbolic representation that we describe later.

THEOREM 2. *The functions $\mathtt{T}^k, \mathtt{P}^k$ satisfy following equations:*

1. $\mathtt{T}^0(o_0, s) = (PI(s) > 0) \wedge (O_i(s) = o_0)$
2. $\mathtt{T}^{k+1}(o_0...o_{k+1}, s) = \exists t \in S : (\mathtt{T}^k(o_0...o_k, t) \wedge (t \rightarrow s) \wedge (O_i(s) = o_{k+1}))$
3. $\mathtt{P}^0(o_0, s) = (O_i(s) = o_0) \times PI(s)$
4. $\mathtt{P}^{k+1}(o_0...o_{k+1}, s) = \sum_{t \in S} \mathtt{P}^k(o_0...o_k, t) \times PT(t, s) \times (O_i(s) = o_{k+1})$

The functions $\mathtt{T}^k_\phi, \mathtt{P}^k_\phi$ satisfy the following:

1. $\mathtt{T}^k_p(o_0...o_k, s) = \mathtt{T}^k(o_0...o_k, s) \wedge (p \in \pi(s))$
2. $\mathtt{T}^k_{\varphi_1 \wedge \varphi_2}(o_0...o_k, s) = \mathtt{T}^k_{\varphi_1}(o_0...o_k, s) \wedge \mathtt{T}^k_{\varphi_2}(o_0...o_k, s)$
3. $\mathtt{T}^k_{\neg\varphi}(o_0...o_k, s) = \mathtt{T}^k(o_0...o_k, s) \wedge \neg\mathtt{T}^k_\varphi(o_0...o_k, s)$
4. $\mathtt{T}^k_{K_i\varphi}(o_0...o_k, s) = \mathtt{T}^k(o_0...o_k, s) \wedge \forall t \in S : (\mathtt{T}^k(o_0...o_k, t) \Rightarrow \mathtt{T}^k_\varphi(o_0...o_k, t))$
5. $\mathtt{T}^k_{L(P_1,...,P_m)\bowtie c}(o_0...o_k, s) = L(C_1(o_0...o_k, s), \ldots, C_m(o_0...o_k, s)) \bowtie c$, *where*

$$C_j(o_0...o_k, s) = \frac{\sum_{s \in S} \mathtt{P}^k_{\varphi_j}(o_0...o_k, s)}{\sum_{s \in S} \mathtt{P}^k(o_0...o_k, s)} \quad \text{if } P_j = \mathtt{Pr}_i\varphi_j,$$

$$C_j(o_0...o_k, s) = \frac{\sum_{s \in S} \mathtt{P}^0_{\varphi_j}(o_0, s)}{\sum_{s \in S} \mathtt{P}^0(o_0, s)} \quad \text{if } P_j = \mathtt{Prior}_i\varphi_j,$$

$$C_j(o_0...o_k, s) = \frac{\sum_{s \in S} \mathtt{P}^k_{\varphi_j \wedge \psi_j}(o_0...o_k, s)}{\sum_{s \in S} \mathtt{P}^k_{\psi_j}(o_0...o_k, s)} \quad \text{if } P_j = \mathtt{Pr}_i(\varphi_j \mid \psi_j), \text{ and}$$

$$C_j(o_0...o_k, s) = \frac{\sum_{s \in S} \mathtt{P}^0_{\varphi_j \wedge \psi_j}(o_0, s)}{\sum_{s \in S} \mathtt{P}^0_{\psi_j}(o_0, s)} \quad \text{if } P_j = \mathtt{Prior}_i(\varphi_j \mid \psi_j).$$

6. $\mathtt{P}^k_\varphi(o_0...o_k, s) = \mathtt{T}^k_\varphi(o_0...o_k, s) \times \mathtt{P}^k(o_0...o_k, s)$

We remark that the expressions for $\mathtt{T}^k_\phi$ contain occurrences of the term $\mathtt{T}^k(o_0...o_k, s)$. In the context of the right hand side of the formula in Theorem 1, these terms always evaluate to 1 and may be eliminated. This is an optimization done in our algorithm.

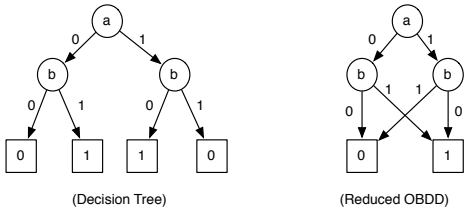

Figure 1: A decision tree and its reduced OBDD

3.1 Symbolic Data Structures

As we noted above, the size of the set of runs of a given length k grows exponentially. The same applies to the tabular representation of the functions $\mathtt{T}^k$ and $\mathtt{P}^k$. Our symbolic algorithm attempts to deal with this exponential growth by representing these functions more compactly than as an input-output table, using two types of data structures: (reduced) ordered binary decision diagrams (OBDD) [3] and multi-terminal binary decision diagrams (MTBDD) [6]. These structures are defined as follows.

Let V be a set of variables. A *V-assignment* is a function $s : V \rightarrow \{0, 1\}$. Write $Assgts(V)$ for the set of all V-assignments, and $s[v \mapsto x]$ for the function that is identical to s except that it takes value x on input v. A *V-indexed boolean function* is a mapping $f : Assgts(V) \rightarrow \{0, 1\}$. Note that such functions are able to represent sets $X \subseteq Assgts(V)$ by their characteristic functions f_X, mapping s to 1 just in case $s \in X$. One way to represent such a function f is using a *binary tree* of height n, with each level corresponding to one of the variables in V, and leaves labelled from $\{0, 1\}$. This tree can in turn be thought of as a finite state automaton on alphabet $\{0, 1\}$. Reduced[1] OBDD's more compactly represent such a function as a *dag* of height n, with binary branching, by applying the usual finite state automaton minimization algorithm. A very simple example of this for the function $f(a, b, c) = a$ `xor` b is illustrated in Figure 1. In some cases, the degree of compaction obtained in the minimal dag representation is considerable. We note that the amount of compaction obtained is sensitive to the variable ordering used, and finding a variable ordering that minimizes the result is NP-hard, though there exist good heuristics, such as *sifting* [24].

Given this minimal representation of V-indexed boolean functions, it is moreover possible to compute (in practice, often in reasonable time) some operations on these functions, by means of algorithms that take as input the reduced OBDD representation of the input functions and returns the reduced OBDD representation of the result. The operations for which this can be done include the following:

- Boolean operations $\wedge, \neg$, defined pointwise on functions. E.g., if $f, g : Assgts(V) \rightarrow \{0, 1\}$, then $f \wedge g : Assgts(V) \rightarrow \{0, 1\}$ is defined by $(f \wedge g)(s) = f(s) \wedge g(s)$.
- Boolean quantification $\exists, \forall$, e.g., if $f : Assgts(V) \rightarrow \{0, 1\}$ and $v \in V$ then $\exists v(f) : Assgts(V \setminus \{v\}) \rightarrow \{0, 1\}$ maps $s \in Assgts(V \setminus \{v\})$ to $f(s[v \mapsto 0]) \vee f(s[v \mapsto 1])$.

(Reduced) MTBDD's generalize (reduced) OBDD's by allowing finitely many real numbers as the leaves, instead of the two values $0, 1$. This gives a compact representation for a *V-indexed function*, i.e.., a mapping $f : Assgts(V) \rightarrow \mathbb{R}$. As with OBDD's, we may perform certain operations on the represented functions directly at the level of their reduced OBDD representations, e.g.,

[1]Since we always reduce, we usually elide this word in what follows.

- Arithmetic operations $+, -, \times, \div$, etc. For instance, if $f, g : Assgts(V) \to \mathbb{R}$, then $f \times g : Assgts(V) \to \mathbb{R}$ is defined by $(f \times g)(s) = f(s) \times g(s)$.

- Summation operation $\sum$. If $f : Assgts(V) \to \mathbb{R}$ and $V' \subseteq V$ then $\sum(f, V') : Assgts(V \setminus V') \to \mathbb{R}$ such that $\sum(f, V')(s) = \sum_{s' \in Assgts(V')} f(s \cup s')$.

- Transformation operation $2bdd$. If $f : Assgts(V) \to \mathbb{R}$ then $2bdd(f) : Assgts(V) \to \{0, 1\}$ such that $2bdd(f)(s) = 1$ if $f(s) \neq 0$ and $2bdd(f)(s) = 0$ otherwise.

Our algorithm for model checking probabilistic knowledge relies on an MTBDD representation of the model M as the input to the model checking problem. To represent relations, we use the set of "primed" versions of the state variables $Prop$, defined by $Prop' = \{v' \mid v \in Prop\}$.

1. The initial probability PI is represented as the MTBDD of the function $f_{PI} : Assgts(Prop) \to [0, 1]$ such that $f_{PI}(s) = PI(s)$.

2. The probabilistic transition relation PT is represented by the MTBDD of the function $f_{PT} : Assgts(Prop \cup Prop') \to [0, 1]$ such that if $s \in Assgts(Prop)$ and $s' \in Assgts(Prop')$, then $f_{PT}(s \cup s') = PT(s, s')$. The transition relation $\to$ is then the function $2bdd(PT)$.

3. Assuming $S \subseteq Assgts(Prop)$, the set of states S can be represented by the MTBDD of its characteristic function $f_S : Assgts(Prop) \to \{0, 1\}$. Furthermore, the observation functions O_i are associated with indistinguishability relations $\sim_i$ on states, defined by $s \sim_i s'$ if $O_i(s) = O_i(s')$. These can be represented as boolean functions $f_{\sim_i} : Assgts(Prop \cup Prop') \to \{0, 1\}$ such that if $s \in Assgts(Prop)$ and $s' \in Assgts(Prop')$, then $f_{\sim_i}(s \cup s') = 1$ iff $s \sim_i s'$. We also assume that each agent i observes some set of variables $Obs_i \subset Prop$, so that $s \sim_i s'$ is equivalent to $s \restriction Obs_i = s' \restriction Obs_i$.

Given this input, the symbolic algorithm proceeds by constructing an MTBDD representation for the universally quantified formula in Theorem 1, using the recurrences of Theorem 2 for terms of the form $\mathrm{T}^k(o_0 \dots o_k, s)$ and $\mathrm{P}^k(o_0 \dots o_k, s)$. Each argument o_j here corresponds to a set of MTBDD variables $Obs_{i,j}$ in one-to-one correspondence to Obs_i, and the argument s corresponds to the variables $Prop$. The recurrences can then each be straightforwardly represented using the MTBDD operations over MTBDD's over the appropriate set of variable.

We give just one example of this representation here, for recurrence 4 of Theorem 2. Suppose we have an MTBDD representation rep_1 of the function $\mathrm{P}^k(o_0 ... o_k, s)$ using variables $(\cup_{j=0...k} Obs_{i,j}) \cup Prop$, representation rep_2 of $PT(s, s')$ using variables $Prop \cup Prop'$, and representation rep_3 of $(O_i(s) = o_{k+1})$ using variables $Prop$. We treat each of these as MTBDD's over the set of variables

$$(\cup_{j=0...k} Obs_{i,j}) \cup Prop \cup Prop' .$$

The encoding of the function $\mathrm{P}^{k+1}(o_0 ... o_{k+1}, s)$ is then the result of the MTBDD expression

$$(\sum (rep_1 \times rep_2 \times rep_3[Prop'/Prop], Prop))[Prop/Prop'],$$

where $rep[Prop/Prop']$ relabels the variables in the MTBDD by replacing each $p' \in Prop'$ by the corresponding variable p, and similarly for the converse operation.

4. IMPLEMENTATION IN MCK

We have implemented the symbolic algorithm describe above as an extension of MCK, a model checker for the logic of knowledge and time [12]. MCK takes as input a script that describes a multi-agent system, in which the behaviour of agents is represented using a simple programming language. The script declares a set of agents, a set of variables shared by the agents, and describes how transitions occur as a function of actions chosen by the agents, by means of a program. Agents choose their actions by executing a protocol, also in the form of a program. This program may declare further variables that are local to the agent executing it. It also indicates which of the local and shared variables are observable to the agent. Finally, the program describes how the agent chooses its actions as a function of these variables at each step of execution. States of the system are derived as assignments to the shared, local and program counter variables. Initial states are described using a boolean formula over these variables.

In order to develop the probabilistic extension of MCK, we replace the *non-deterministic if* construct in the MCK programming language by a *weighted if* construct. The syntax of the latter is of the form

$$\textbf{if } w_1 : \phi_1 \to P_1 \ [] \ w_2 : \phi_2 \to P_2 \ [] \ \dots \ [] \ w_k : \phi_k \to P_k \textbf{ fi}$$

where the w_i are rational numbers, the ϕ_i are boolean expressions and the P_i are programs. If all weights are equal, the prefix "w_i :" may be elided. Intuitively, this construct corresponds to a probabilistic transition to the programs P_i, and is executed from state s as follows. Let $w = \sum_{i=1...k,\ s \models \phi_i} w_i$. Then if $s \not\models \phi_i$, we make a transition to P_i with probability 0, otherwise we make the transition to P_i with probability w_i/w. (If all ϕ_i are false, we skip over the statement.)

The formal semantics of the extended MCK scripts construct a PO-DTMC from the input script. Examples of input scripts in the extended language are given in the following section.

The implementation of the MTBDD-based model checking algorithm described above uses the MTBDD capabilities of the CUDD package [**?**].

5. APPLICATIONS

To investigate the application of the algorithm described above, we have used the MCK implementation to verify several security properties in two protocols: Chaum's Dining Cryptographers protocol [5] and an oblivious transfer protocol due to Rivest [23]. These protocols have previously been the subject of analysis by epistemic model checking (starting with [28], which uses an algorithm that we have generalized in this paper), so it is interesting to compare the epistemic versions (which use only on the operator K_i) with the probabilistic approach of the present paper. We report performance results for our implementation. All experiments in the paper were conducted on an iMac computer with 3.06GHz Intel Core i3 processor and 4GB memory.

5.1 Oblivious Transfer Protocol

The objectives of an *oblivious transfer* protocol are as follows: Alice has two secrets m_0 and m_1. Bob would like to learn one of these secrets, but does not wish to reveal to Alice which secret he chooses. Various approaches have been proposed by which this may be achieved. In Rivest's solution [23], a trusted third party Ted is used, who helps Alice and Bob by providing some random material, that they use once they decide to run the protocol. (The point is that Ted's participation is not required at runtime: if this were allowed then there is of course a trivial solution in which Alice

sends Ted her messages, and Bob requests the message he wants from Ted.) The random material consists of two random strings r_0, r_1 of the same length as Alice's messages, and a random bit d. Identical copies of these values are delivered by Ted to both Alice and Bob. To request a string c, Bob then sends Alice the value $e = c \oplus d$ (where $\oplus$ is the exclusive-or operation, which we assume operates pointwise on strings of equal length.). Alice responds with the messages $f_0 = m_0 \oplus r_e$ and $f_1 = m_1 \oplus r_{1-e}$. From this information, Bob is able to compute m_c as $f_c \oplus r_d$, but this exchange leaves Alice ignorant of which message Bob chose, and leaves Bob ignorant of the value of Alice's other message.

The following is precise description of the protocol in the MCK programming language for the case where Alice's message has length 2 bits. We first declare the variables required and their types. The expression "`Bool[n]`" denotes the type of Boolean vectors of length n. Comments are prefixed by "`--`".

```
-- Alices two messages:
m0: Bool[2]
m1: Bool[2]

-- c is Bob's choice of message to receive
c: Bool

-- Bob's reception buffer
mc: Bool[2]

-- Random numbers generated by Ted
r0 : Bool[2]
r1 : Bool[2]
d : Bool

-- rd = if d then r1 else r0
rd : Bool[2]

-- a value computed by Bob and sent to Alice
e : Bool
-- values computed by Alice and sent to Bob
f0 : Bool[2]
f1 : Bool[2]
```

Since not all assignments to these variables constitute a valid initial state, we next define the valid initial states as all states satisfying a formula over these variables:

```
init_cond =
-- (1) rd = if d then r1 else r0
(neg d =>((r0[0] <=> rd[0]) /\ (r0[1] <=> rd[1])))
/\ (d =>((r1[0] <=> rd[0]) /\ (r1[1] <=> rd[1])))
--  (2) not m0=m1
/\ (neg (m0[0] <=> m1[0]) \/ neg (m0[1] <=> m1[1]))
```

The semantics of this construct places a uniform distribution over the set of global states that satisfy the initial condition.

We then declare the agents in the system:

```
agent Alice "alice" (r0, r1, m0, m1, f0, f1, e)
agent Bob "bob" (rd, d, c, f0, f1, mc)
```

Each of these statements first gives the name of the agent, then (in quotes) the name of the protocol being executed by the agent, and then lists the variables to which that protocol has access. For example, the above statements say that Alice has access to variables `r0` and `r1`, but Bob does not.

MCK splits the description of state transitions into two parts: in each step, each agent's protocol is used to select an action for each agent (this choice may be probabilistic, using the weighted `if` statement introduced above). These choices are then provided as input to the *transitions* clause, another program that computes the values of global variables in the next state from the values in the current state and the agent's actions. This program may also use the weighted `if` statement. In our modelling of Rivest's protocol, most of the work is done in the transitions clause rather than in the agent's protocols:

```
transitions
begin
if  Bob.SendRequest -> e := d xor c
-- Alice sends f0 = m0 xor r_e and f1 = m1 xor r_{1-e}
[]  Alice.SendMessages /\ neg e -> begin
        f0[0]:= m0[0] xor r0[0];
        f0[1]:= m0[1] xor r0[1];
        f1[0]:= m1[0] xor r1[0];
        f1[1]:= m1[1] xor r1[1]
        end
[]  Alice.SendMessages /\ e -> begin
        f0[0]:= m0[0] xor r1[0];
        f0[1]:= m0[1] xor r1[1];
        f1[0]:= m1[0] xor r0[0];
        f1[1]:= m1[1] xor r0[1]
        end
-- Bob computes mc = f_c xor rd
[]  Bob.Compute /\ neg c -> begin
        mc[0]:= f0[0] xor rd[0];
        mc[1]:= f0[1] xor rd[1]
        end
[] Bob.Compute /\ c -> begin
        mc[0]:= f1[0] xor rd[0];
        mc[1]:= f1[1] xor rd[1]
        end
fi
end
```

Here `Alice.SendMessages` is a proposition indicating that Alice has chosen to perform her action `SendMessages` in this step. Superficially, the `transitions` clause uses the weighted `if` statement, but we note that since the conditions prove to be mutually exclusive in all runs, the actual probabilities in the model come entirely from the distribution over the initial states.

The agents' protocols simply select the appropriate action depending on the stage of the protocol. In the first step Bob performs the action `Sendrequest`, next Alice performs `SendMessages`, and finally Bob performs `Compute`.

```
protocol "alice" (
r0 : observable Bool[2],
r1: observable Bool[2],
m0 : observable Bool[2],
m1: observable Bool[2],
f0 : observable Bool[2],
f1: observable Bool[2],
e: observable Bool
)

begin
skip; << SendMessages >>; skip
end

protocol "bob" (
rd: observable Bool[2],
d: observable Bool,
c: observable Bool,
f0: observable Bool[2],
f1: observable Bool[2],
mc: observable Bool[2])
begin
<< SendRequest >>; skip; << Compute >>
end
```

The notation "`<< . >>`" is used to signify an action. The label `observable` is used to indicate that a variable is a part of the agent's observation in a global state.

(1)	$X^3((c = 0 \Rightarrow \mathtt{Pr}_{\mathrm{Bob}}(mr = m0) = 1) \wedge (c = 1 \Rightarrow \mathtt{Pr}_{\mathrm{Bob}}(mr = m1) = 1)$
(2)	$X^3(\mathtt{Pr}_{\mathrm{Alice}}(c = 0) = \mathtt{Pr}_{\mathrm{Alice}}(c = 1))$
(3)	$X^3((c = 1) \wedge (m1 \neq \mathtt{m}') \wedge (m1 \neq \mathtt{m}'') \Rightarrow (\mathtt{Pr}_{\mathrm{Bob}}(m0 = \mathtt{m}') = \mathtt{Pr}_{\mathrm{Bob}}(m0 = \mathtt{m}''))$ (for *constant* messages $\mathtt{m}', \mathtt{m}''$)

Table 2: Properties of the Oblivious Transfer Protocol

The objectives of the protocol can be captured by the formulas in Table 2. Formula (1) says that Bob knows that the message `mr` (which the protocol uses to represent the message that Bob receives) is equal to the message that he requested. Formula (2) says that from Alice's point of view, the probability of either choice `c` by Bob is the same. Formula (3) says that if $c = 1$, i.e., Bob chooses to receive `m1`, then Bob does not learn the value of the other message $m0$. Since it is assumed that `m0`≠`m1`, Bob does know that `m0` is not equal to the message `m1` he has received. The formula says that from Bob's point of view, if he chooses message `m1`, then Bob's considers no fixed value $\mathtt{m}'$ of `m0` more probable than any other fixed value $\mathtt{m}''$. (Appealing to symmetry, we model check this for just two possible fixed values $\mathtt{m}', \mathtt{m}''$.)

We have used our implementation to verify that the protocol satisfies all three formulas. To test how well our algorithm scales, we have performed experiments in which we parameterize the protocol on the length of messages `m0`,`m1`, represented as Boolean vectors of length n. For our modelling, the state-space of the protocol itself grows as 2^{7n+c} (in the initial state, there are 8 boolean vectors of length n, but the value of `rd` is dependent) so that the expected performance of a naive algorithm that explicitly generates the entire state-space in order to evaluate the formulae would be exponential.

Figure 2 plots the runtimes for specification (1) in a base 2 log-scale. For purposes of comparison with non-probabilistic epistemic model checking, we also plot the runtimes obtained using the algorithm of [28] on the epistemic logic formulation

$$X^2((c = 0 \Rightarrow K_{\mathrm{Bob}}(mr = m0)) \wedge (c = 1 \Rightarrow K_{\mathrm{Bob}}(mr = m1))$$

on a variant of the model in which the initial probabilities are ignored. In both cases, the sifting optimization was used during BDD construction (invoked in MCK with the flag `-rs`). Fitting a straight line to the data yields an approximation of $y = 0.24x + 1.2$ (with norm of residuals 1.7) for the purely epistemic specification and $y = 0.24x + 2$ (with norm of residuals 1.9) for the probabilistic specification (1). Compared to the expected curve of at least $y = 7x + c$ for a brute force model checking approach based on explicit generation of all states and traces, this suggests that our symbolic model checking approach yields a significant optimization that brings problems of considerable scale into the range of feasibility. (Note that the largest of our examples involve more than $2^{7\times35} = 5.7\times10^{73}$ states.) Interestingly, in this example, the addition of probabilistic model checking capability has come at only a small cost over the cost of epistemic model checking. The runtimes for all three specifications together turns out to be only slightly larger than those for just formula (1), since much of the MTBDD construction is shared between specifications.

5.2 The Dining Cryptographers Protocol

The Dining Cryptographers protocol [5] is intended to allow anonymous message transmission. Chaum introduced it by the following story. Three cryptographers are sitting down to dinner at their favorite three-star restaurant. Their waiter informs them that arrangements have been made 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. In order to resolve this question, they devise a protocol that operates as follows: each flips a coin, and privately shares the outcome with their neighbour to the right around the (circular) table. They then each publicly announce whether the outcome of their own coin and the coin shared by their neighbour to the left are the same, except that if they paid for the dinner, they invert their answer. The answer to the question about the NSA can then be determined from the exclusive-or of the public announcements, and it can be shown that in case one of the cryptographers paid, their identity is not revealed to the others. The idea generalizes to n cryptographers in a ring (indeed, to connected graphs of cryptographers). In the following, we assume n cryptographers in a ring and let $i, j, k \in \{1, \ldots, n\}$.

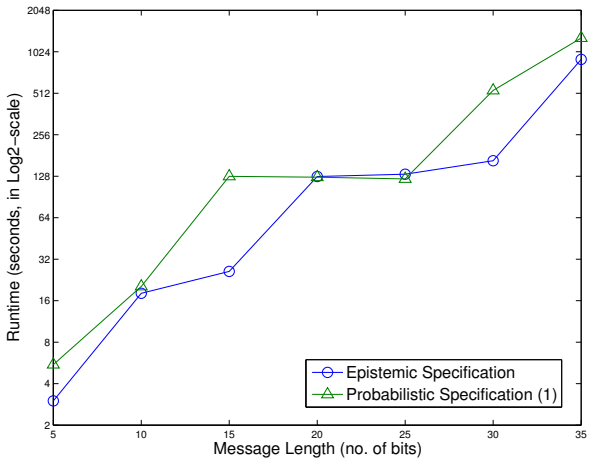

Figure 2: Oblivious Transfer Protocol, Runtime

The anonymity property has been represented in previous work on epistemic analysis of the protocol using the formula (1) of Table 3 which says that, in case agent i pays, agent $j \neq i$ cannot eliminate the possibility that any other agent $k \notin \{i, j\}$ paid. Here d is the number of steps taken to complete the protocol (which depends on the details of the modelling).

While this formula gives some interesting information about the protocol, it does not provide a completely satisfactory analysis from the point of view of security. For example if cryptographer 1 ascribes probability 0.999 to $paid_2$ and probability 0.001 to $paid_3$, then both are possible, but for all practical purposes it is as good as known that $paid_2$. Chaum, indeed, shows that the protocol satisfies a stronger probabilistic anonymity property.

(1)	$X^d(paid_i \Rightarrow (\bigwedge_{k \notin \{i,j\}} \neg K_j(\neg paid_k)))$	(4)	$X^d(paid_i \Rightarrow \bigwedge_{k \notin \{i,j\}}(\mathtt{Pr}_j paid_i \leq \mathtt{Pr}_j paid_k))$
(2)	$X^d(paid_i \Rightarrow \mathtt{Pr}_j paid_i \leq 1 - \epsilon)$	(5)	$X^d((K_j \bigvee_{i \neq j} paid_i) \Rightarrow \bigwedge_{i \neq j}(\mathtt{Pr}_j paid_i = \mathtt{Prior}_j(paid_i \mid \bigvee_{k \neq j} paid_k)))$
(3)	$X^d(paid_i \Rightarrow \mathtt{Pr}_j paid_i \leq 0.5)$		

Table 3: Anonymity Properties of the Dining Cryptographers Protocol

In the subsequent literature [22, 14] on anonymity protocols, several different probabilistic definitions of anonymity have been proposed:

1. A sender is *possibly innocent* if, from the attacker's point of view, there is a nontrivial probability that the real sender is someone else. For the DC protocol we can express this by the formula (2) of Table 3, where ϵ is a small positive number.
2. A sender is *probably innocent* if, from the attacker's point of view, the sender appears no more likely to be the originator of the message than to not be the originator. The formula for this is (3).
3. A sender is *beyond suspicion* if, though the attacker can see the evidence of a sent message, the sender appears no more likely to be the originator of the message than any other potential sender in the system. This can be expressed as formula (4).
4. *Conditional Anonymity* [14] in the cryptographers protocol can be characterized by formula (5). Intuitively, this formula says that all though the protocol reveals whether one of the cryptographers paid, that is all the new information that any cryptographer learns about who is the culprit: the probability ascribed to the payer being the culprit is not increased over i's prior for this any more than follows from just this new knowledge. (Note that the protocol satisfies $X^d((K_j \bigvee_{i \neq j} paid_i) \vee K_j \bigwedge_{i \neq j} \neg paid_i)$.)

Besides these different notions of anonymity, the literature has considered the effect of deviations from uniform probability distributions [4]. There are two sources of probability in the protocol: the prior probability of any of the cryptographers being the payer (which might, e.g., assign a higher probability to rich Adi over poor David), and the coin tosses.

We use the notation $w_1 : w_2 : \ldots : w_k$ to denote the distribution in which the i-th component has probability $w_i/(w_1 + \ldots w_k)$. thus, a prior of $n : 1 : 1 : 1 : 1$ means that, in a protocol of 4 cryptographers, with probability $\frac{n}{n+4}$, the cryptographer 1 pays, with probability $\frac{1}{n+4}$, other agents pay, and with probability $\frac{1}{n+4}$, no one pays. A coin might be fair or unfair: a fair coin gives a half-to-half chance for each side, while a unfair coin gives different chances for heads and tails. A $n:1$ coin means that there is probability $\frac{n}{n+1}$ that it lands heads and probability $\frac{1}{n+1}$ that it lands tails.

We have experimentally studied the effectiveness of our algorithm for checking these properties. Figure 3 compares the runtimes, for different numbers of cryptographers, for the nondeterministic specification (1) solved by the algorithm proposed in [28], with those for the probabilistic specification (2), with fair coins, solved by the algorithm of the present paper. (Very similar runtimes are found for the other probabilistic specifications.) The results indicate that the move from nondeterministic to a probabilistic model checking does come at the cost of increased runtime and a steeper slope. Both approaches appear to grow at an exponential rate, so in the case of this protocol it is less clear that the symbolic approach has yielded significant performance benefits. (However, [28] shows that an alternate modelling that points to possible optimizations of the algorithm of that paper allows larger numbers of agents to be more efficiently handled, and we expect that similar results could be obtained for the probabilistic case.)

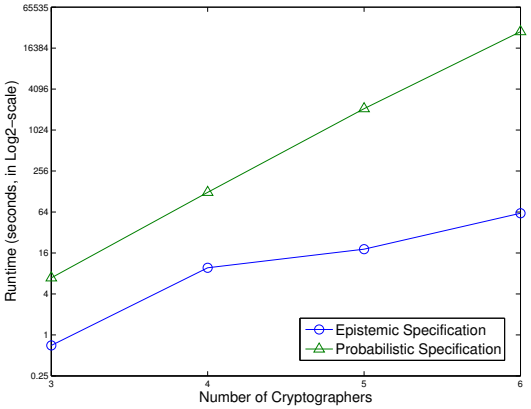

Figure 3: Dining Cryptographers Protocol, Runtime

Table 4 shows the experimental results on the truth of anonymity properties (2)-(5) on probabilistic dining cryptographer protocols, where ϵ is fixed at 0.1 for formula (2). We can see that, beyond suspicion is the most vulnerable anonymity degree in that small changes to the parameters (i.e., coins or priors) as described would break it. For the possible innocence and probable innocence, they will eventually be broken, with the change of either of the parameters of the protocol. Between them, the probable innocence is more vulnerable. This result corresponds with the observation in [22].

Table 5 shows the experimental results on the truth of formula (2) with the different value of ϵ. As the degree of bias in either the prior or the coins increases, we see that we need to decrease ϵ to smaller numbers in order for possible innocence to hold.

6. RELATED WORK

There already exist a number of probabilistic model checkers based on symbolic techniques. Most notably, PRISM [16], based on OBDD and MTBDD, has implemented a comprehensive set of symbolic model checking algorithms on various probabilistic models of complete information. The difference with our work is that PRISM, in effect, works just with the prior measure μ_R on runs. To express agent knowledge in PRISM, it is necessary to specify a formula representing a *specific* observation-sequence on which to condition the prior, whereas our approach has an implicit quantification over *all* possible observation sequences, and updates agent knowledge to the current time.

Protocols		Anonymity Degrees			
Coin	Prior	(2)	(3)	(4)	(5)
1:1	1:1:1:1:1	Y	Y	Y	Y
2:1	1:1:1:1:1	Y	Y		
3:1	1:1:1:1:1	Y			
11:1	1:1:1:1:1				
1:1	2:1:1:1:1	Y	Y		Y
1:1	3:1:1:1:1	Y			Y
1:1	19:1:1:1:1				Y

Table 4: Anonymity Properties ($\epsilon = 0.1$)

Protocols		Formula (2)		
Coin	Prior	$\epsilon = 0.5$	$\epsilon = 0.05$	$\epsilon = 0.005$
1:1	1:1:1:1	Y	Y	Y
2:1	1:1:1:1		Y	Y
20:1	1:1:1:1			Y
200:1	1:1:1:1			
1:1	2:1:1:1		Y	Y
1:1	20:1:1:1			Y
1:1	200:1:1:1			

Table 5: Possible Innocence of different ϵ

Partially observable Markov decision processes (POMDP) [18], which generalize PO-DTMC, have been widely studied in AI, but the focus is generally on heuristic approaches to the discovery of optimal strategies that can be taken by an agent. *Belief space* in this area corresponds to agent's perfect recall subjective probability, but is generally represented explicitly rather than symbolically. This area also does not focus on the general class of specifications we consider. Towards model checking both probability and knowledge, [8] works on a logic where the semantics of knowledge and probability are orthogonal and the knowledge is interpreted on current observation. However, the temporal dimension and perfect recall semantics in our work are not considered.

Security properties like those in the protocols we study have been previously considered from the perspective of both probabilistic epistemic logic and probabilistic model checking. To analyze anonymity protocols like the dining cryptographers protocol, [14] proposes some properties based on the probabilistic knowledge of the agents. We have shown in the paper that our algorithm can verify formulas expressing those properties. A different approach to specification of anonymity is taken in [4], which analyzes the Dining cryptographers protocol by concentrating on the conditional probability $P(o|a)$, expressing the probability of observations o under the condition of system behaviors a. PRISM has been used to analyze some security protocols, e.g., the crowds protocol [25] and probabilistic contract signing protocols [21]. These analyses concentrate on the objective aspects of the system, instead of the subjective view of agents as in this paper.

7. CONCLUSIONS

We have shown in this paper that model checking perfect recall probabilistic knowledge is feasible in at least some simple examples. This is just a first step, and we believe that much can be done to optimize the performance of our algorithm by eliminating some obvious redundancies prior to MTBDD construction. Another interesting direction is to broaden the scope of the formulas that can be handled, to encompass a richer temporal expressiveness. We plan to pursue these directions in future research.

8. REFERENCES

[1] O. I. Al-Bataineh and R. van der Meyden. Abstraction for epistemic model checking of dining cryptographers-based protocols. In *TARK XIII*, 2011.

[2] K. Baukus and R. van der Meyden. A knowledge based analysis of cache coherence. In *Lecture Notes in Computer Science*, volume 3308, pages 99–114, 2004.

[3] R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

[4] K. Chatzikokolakis, C. Palamidessi, and P. Panangaden. Anonymity protocols as noisy channels. *Information and Computation*, 206(2-4):378–401, 2008.

[5] D. Chaum. The dining cryptographers problem: unconditional sender and recipient untraceability. *J. Cryptol.*, 1:65–75, March 1988.

[6] E. Clarke, M. Fujita, P. McGeer, K. McMillan, J. Yang, and X. Zhao. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. *Formal Methods in System Design*, 10(2):149–169, 1997.

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

[8] C. A. D. M. Delgado and M. R. F. Benevide. Verification of epistemic properties in probabilistic multi-agent systems. In *LNCS 5774*, pages 16–28, 2009.

[9] D. Eijck. Dynamic epistemic modelling. *CWI. Software Engineering [SEN]*, (E 0424):1–112, 2004.

[10] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning About Knowledge*. The MIT Press, 1995.

[11] R. Fagin and J. Y. Halpern. Reasoning about knowledge and probability. *J. ACM*, 41:340–367, March 1994.

[12] P. Gammie and R. van der Meyden. MCK: Model checking the logic of knowledge. In *Proc. Conf. on Computer-Aided Verification, CAV*, pages 479–483, 2004.

[13] J. Y. Halpern. *Reasoning about Uncertainty*. MIT Press, 2003.

[14] J. Y. Halpern and K. R. O'Neill. Anonymity and information hiding in multiagent systems. *Journal of Computer Security*, 13(3):483–514, 2005.

[15] J. Y. Halpern and M. R. Tuttle. Knowledge, probability, and adversaries. *Journal of the ACM*, 40:917–960, 1993.

[16] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 441–444. Springer, 2006.

[17] M. Kacprzak, W. Nabiałek, A. Niewiadomski, W. Penczek, A. Półrola, M. Szreter, B. Woźna, and A. Zbrzezny. Verics 2007 - a model checker for knowledge and real-time. *Fundamenta Informaticae*, 85(1):313–328, 2008.

[18] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

[19] A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: a model checker for the verification of multi-agent systems. In *Proc. Conf. on Computer-Aided Verification*, pages 682–688, 2009.

[20] N. O. G. Nikolay V. Shilov. Model checking knowledge and fixpoints. In *Fixed Points in Computer Science*, pages 25–39, 2002.

[21] G. Norman and V. Shmatikov. Analysis of probabilistic contract signing. *Journal of Computer Security*, 14(6):561–589, 2006.

[22] M. K. Reiter and A. D. Rubin. Crowds: anonymity for web transactions. *ACM Trans. Inf. Syst. Secur.*, 1(1):66–92, 1998.

[23] R. L. Rivest. Unconditionally secure commitment and oblivious transfer schemes using private channels and a trusted initializer. Unpublished manuscript, 1999.

[24] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *1993 IEEE/ACM international conference on Computer-aided design*, 1993.

[25] V. Shmatikov. Probabilistic model checking of an anonymity system. *Journal of Computer Security*, 12(3-4):355–377, 2004.

[26] K. Su, A. Sattar, and X. Luo. Model checking temporal logics of knowledge via OBDDs. *The Computer Journal*, 50(4):403–420, 2007.

[27] R. van der Meyden and N. V. Shilov. Model checking knowledge and time in systems with perfect recall. In *LNCS 1738*, pages 432–445, 1999.

[28] R. van der Meyden and K. Su. Symbolic model checking the knowledge of the dining cryptographers. In *17th IEEE workshop on Computer Security Foundations*, pages 280–291, 2004.

[29] H. van Ditmarsch, W. van der Hoek, R. van der Meyden, and J. Ruan. Model checking Russian cards. In *ENTCS 149(2)*, pages 105–123, 2006.