Probabilistic Modal Logic

Afsaneh Shirazi and **Eyal Amir**
Computer Science Department, University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
{hajiamin, eyal}@uiuc.edu

Abstract

A modal logic is any logic for handling modalities: concepts like possibility, necessity, and knowledge. Artificial intelligence uses modal logics most heavily to represent and reason about knowledge of agents about others' knowledge. This type of reasoning occurs in dialog, collaboration, and competition. In many applications it is also important to be able to reason about the probability of beliefs and events.

In this paper we provide a formal system that represents probabilistic knowledge about probabilistic knowledge. We also present exact and approximate algorithms for reasoning about the truth value of queries that are encoded as *probabilistic modal logic* formulas. We provide an exact algorithm which takes a probabilistic Kripke structure and answers probabilistic modal queries in polynomial-time in the size of the model. Then, we introduce an approximate method for applications in which we have very many or infinitely many states. Exact methods are impractical in these applications and we show that our method returns a close estimate efficiently.

1 Introduction

The study of knowledge in artificial intelligence is both theoretical and applied. Answers to "what do we know?", "what can be known?", and "what does it mean to say that someone knows something?" apply to many important areas (Halpern 1987; Fagin *et al.* 1995; Friedman & Halpern 1994; Aumann 1986). Formal models of reasoning about knowledge use modal operators and logic to express *knowledge* and *belief*. These enable discussion of knowledge of logical expressions (e.g., *James does not know that Jill is at home*) separately from the truth values of these expressions (it holds that *Jill is at home*).

In many applications, it is important to consider knowledge more gradually. For example, a poker player may not know what his opponents' cards are, but may have a probability distribution over the possible hands. Moreover, he should have an estimate of other players' estimates of his hand, or otherwise he cannot bluff effectively. Current formal logical systems (especially modal logics) (Fitting 1993) cannot represent this knowledge or simulate it. On the other hand, probabilistic representations (Pearl 1988) can represent distributions over distributions (e.g., Dirichlet priors over multinomial distributions (Blei, Ng, & Jordan 2003)), but are not granular enough for multiple levels of complex beliefs about beliefs. Furthermore, reasoning with these representations is computationally hard because they mix structure, continuous variables, and discrete variables (necessary for distributions over distributions).

In this paper we address the need for granular representation of probabilistic belief about probabilistic beliefs. We develop and present a representation and reasoning algorithms for nested probabilistic modalities. We describe syntax, semantics, and tractable algorithms for reasoning with our representation. Our reasoning algorithms evaluate the value of a query on a state given a model.

On the way to these contributions we provide a theory for *probabilistic modal logic*. We introduce a framework for modeling probabilistic knowledge that is based on modal logic and possible worlds semantics with a probability distribution over the accessibility relations. We introduce two exact methods (Top-Down and Bottom-Up) that we can choose from based on the properties of our application. We show that when the number of nested modal functions is small, our Top-Down method works faster, whereas when we have complex nesting the Bottom-Up approach (which is polynomial-time in the number of states) is faster. For very large (even infinite) models neither of these methods is tractable. Therefore, we use sampling to estimate the truth value of the query. In our approximation method we reduce our problem to inference in Bayesian networks, so all the exact and variational methods applicable to Bayesian networks can be used.

Most previous related works are limited to combining probability with a special case of modal logic in which accessibility relations are equivalence relations (we call this special case *probabilistic knowledge*). Among those, (Fagin & Halpern 1988; Heifetz & Mongin 1998) are mainly concerned with providing a sound and complete axiomatization for the logic of knowledge and probability. In contrast, we focus on providing tractable reasoning methods to answer queries (on one model) for the general probabilistic modal logic, as well as the special case of probabilistic knowledge.

Another work related to ours is (Milch & Koller 2000) in which probabilistic epistemic logic is used to reason about the mental states of an agent. This logic is a special case of probabilistic knowledge with the additional assumption of

agents having a common prior probability distribution over states of the world.

Adding probabilistic notions to modal logic is also considered in (Herzig 2003; Nie 1992). The former adds a unary modal operator expressing that a proposition is more probable than its negation, whereas the latter defines an extension of fuzzy modal logic to perform probabilistic semantic-based approaches for finding documents relevant to a query.

2 General Probabilistic Modal Logic

In this section, we present the syntax and semantics of probabilistic modal logic. Our framework is based on probability distributions over possible states of the world. From an agent perspective besides the true state of the world, there are a number of possible states that he cannot distinguish from each other. An agent has a probability distribution over the set of states he considers possible. An agent is then said to *know* a fact φ with probability p if the sum of probabilities of the states in which φ is true is p.

An application is modeling a Holdem game (Texas Holdem poker). In Holdem, players receive two downcards as their personal hand, after which there is a round of betting. Three boardcards are turned simultaneously and another round of betting occurs. The next two boardcards are turned one at a time, with a round of betting after each. A player may use any five-card combination from the board and personal cards.

The poker ranking hands are as follows (from the highest to the lowest). *Straight flush*: five consecutive cards of the same suit. ("Suits" are spades, hearts, clubs and diamonds.) *Four of a kind*: four cards of the same rank. Higher ranks are better (four tens would beat four sixes). *Full house*: three cards of the same rank, with a pair of another rank. *Flush*: five cards of the same suit. *Straight*: five sequential cards of different suits. *Three of a kind*: three cards of the same rank. *Two pair*: two cards of one rank, and two cards of another rank. *One pair*: two cards of the same rank, and three unrelated cards. *No pair, high card*: a hand with no pair or any of the other ranking values above. When comparing no pair hands, the highest card determines the winner.

Suppose that in a two player game, the boardcards are KKQ32, Player 1 has AK, and player 2 has K3. From the perspective of player 1, player 2 can have any card except the boardcards and the cards in player 1's hand. In Holdem, the possible worlds are all the possible ways the cards could have been distributed among the players. Initially, a player may consider possible all deals consistent with the cards in her hand with equal probability. Players may acquire additional information in the course of the game that allows them to eliminate some of the worlds they consider possible or change the probability distribution of others.

To be able to infer some information about the knowledge of an agent, we first need a language that allows us to express notions of knowledge and its probability in a straightforward way. Modal logic is a model for knowledge but it does not contain the probability distribution over different states. For this purpose we introduce *probabilistic modal logic*.

2.1 Syntax

For simplicity we assume that the agent wish to reason about a world that can be described in terms of a nonempty set $\mathcal{P}$ of primitive propositions. Probabilistic modal formulas are built up from a countable set of propositional letters $\mathcal{P}$ using propositional connectives ($\neg$, $\wedge$) and the modal function K. We use $\top$ and $\bot$ for truth and falsehood constants. The formation rules are:

1. Every propositional letter is a formula.
2. $\top$ and $\bot$ are formulas.
3. If X is a formula so is $\neg X$.
4. If X and Y are formulas so is $X \wedge Y$.
5. If X is a formula, $0 < r \leq 1$, and $\alpha \in \{<, =\}$ then $(K(X)\ \alpha\ r)$ is a formula.

All comparison operators can be implemented with rules 3, 4, and 5. Note that we have a different modal function K_i for each agent i in the domain. We take $\vee$, $\supset$ and $\equiv$ to be abbreviations in the usual way.

Suppose that we have two propositional symbols in our Holdem example, w_1 and w_2, to show whether player 1 or player 2 wins the hand, respectively. The value of these symbols is determined based on the game rules applied on players' hands and the boardcards. In this example there are two players, therefore we have two modal functions, K_1 and K_2 corresponding to player 1 and player 2. $K_1(w_1) < 1/2$ is an example of a formula in probabilistic modal logic whose truth value can be evaluated on the current state of the world. $K_1(w_1) < 1/2$ shows that the probability of player 1 winning the hand is less than $1/2$ from her perspective.

2.2 Semantics

In this section we describe the semantics of our language. Our approach is in terms of possible worlds, which is similar to *Kripke structures* in modal logic.

Definition 2.1 *A* probabilistic Kripke structure M *is a tuple* (S, P, V) *in which*

1. S *is a nonempty set of* states *or possible worlds.*
2. P *is a* conditional probability function. $P(s'|s)$ *shows the probability of accessing state* s' *given that we are in state* s. $P(s'|s)$ *greater than zero indicates that* s' *is accessible from* s. *Therefore, it is similar to the accessibility relation of modal logic. Since* P *is a probability function, we should ensure that the following constraints hold:*
   - $0 \leq P(s'|s) \leq 1$
   - *For each state* $s \in S$: $\sum_{s' \in S} P(s'|s) = 1$
3. V *is an* interpretation *that associates with each state in* S *a truth assignment to the primitive propositions* $\mathcal{P}$ *(i.e.,* $V : S \times \mathcal{P} \rightarrow \{$*true, false*$\}$*).*

The above definition is for one modal function. For cases in which we have i modal functions $(K_1, \ldots K_i)$ we need a conditional probability function P_j for each modal function K_j. Intuitively, the conditional probability function P_j represents the probability of transition from one state to the other from the perspective of agent j.

In modal logic, the true state of the world is a state in S, same as in probabilistic modal logic. Based on the true state of the world an agent has a probability distribution over all

the states that are indistinguishable to her. The truth value of any formula is evaluated with the following definitions.

Definition 2.2 *Let $M = (S, P, V)$ be a probabilistic Kripke structure. Let s be a state in S. For any probabilistic modal logic formula φ, we define $val(s, \varphi)$ recursively as follows.*

1. $val(s, R) = V(s, R)$ *for a propositional letter* R.
2. $val(s, \top) = true$.
3. $val(s, \bot) = false$.
4. $val(s, \neg X) = \neg val(s, X)$.
5. $val(s, X \wedge Y) = val(s, X) \wedge val(s, Y)$.
6. $val(s, K(X)\ \alpha\ r) = true$ *iff* $\sum_{s' \in S, val(s', X) = true} P(s'|s)\ \alpha\ r$.

Now we can use val definition to define $\models$.

Definition 2.3 $M, s \models X$ *iff* $val(s, X)$ *is true.*

3 Probabilistic Knowledge (Special Case)

In this section we focus on a special case of probabilistic modal logic in which accessibility relation is an indication of knowledge. We take an accessibility relation to be an equivalence relation since we want to capture the intuition that agent i consider state t accessible from state s if in both s and t agent i has the same information about the world, that is, the two worlds are indistinguishable to the agent. This constraint seems natural and it turns out to be the appropriate choice for many applications (e.g., Texas Holdem poker).

The syntax is the same as probabilistic modal logic.

Definition 3.1 *A* probabilistic knowledge structure M *is a tuple* (S, R, P, V) *in which*

1. S *is a nonempty set of* states *or possible worlds.*
2. $R \subseteq S \times S$ *is a binary relation on* S *called the* accessibility relation. R *is an equivalence relation.*
3. P *is a* probability function *that associates with each state in* S *a number. This number represents the probability of being in that state given that we know that we are in one of the states accessible from* s.
4. V *is an* interpretation *that associates with each state in* S *a truth assignment to the primitive propositions* $\mathcal{P}$.

As in previous section for the cases in which we have more than one modal function (e.g., $K_1, \ldots K_i$) we need a probability function and an accessibility relation corresponding to each modal function. Intuitively, the probability function P_j represents the probability of being in a state from the perspective of agent j.

Based on above definition, all the states in one equivalence class are indistinguishable to the agent. An actual state of the world is a state in S. Since the agent cannot distinguish the states in one equivalence class, the probability of being in each state is computed. The probabilistic knowledge semantics can easily be transformed to probabilistic modal logic semantics.

Definition 3.2 *Let $M' = (S', R', P', V')$ be a probabilistic Knowledge structure. The probabilistic Kripke structure $M = (S, P, V)$ is equivalent to M' when:*

$S = S'$, $V = V'$, *and* $P(s'|s) = \dfrac{P'(s')}{\sum_{s'' \in S, sR's''} P'(s'')}$.

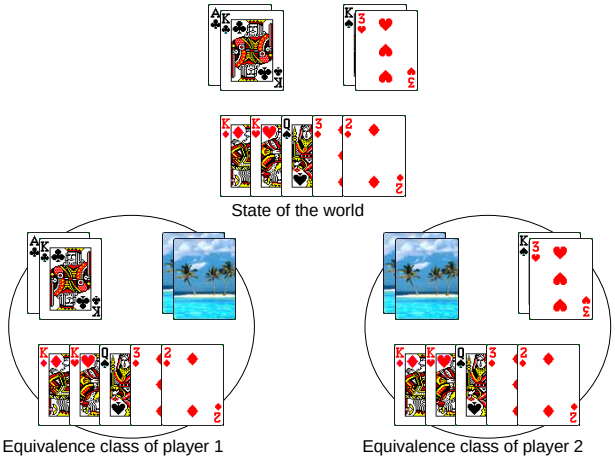

Figure 1: Holdem example equivalence classes.

This formalism is compatible with (Fagin & Halpern 1988). We can model many games with this framework. A good example is Holdem. Suppose that in a two player Holdem, the boardcards are KKQ32, player 1 has AK, and player 2 has K3. We model this with probabilistic knowledge structure $M = (S, R_1, R_2, P_1, P_2, V)$. S is the set of all states, that is, the set of all possible combination of player hands when the boardcards are KKQ32.

R_1 and R_2 are equivalence relations. All the states that are indistinguishable from the perspective of player 1 are in the same equivalence class in R_1. We represent a state as a tuple $(KKQ32, AK, K3)$ in which the components are the boardcards, player 1's hand, and player 2's hand, respectively. Since player 1 does not know anything about her opponent's hand, $(KKQ32, AK, K3)$ and $(KKQ32, AK, 63)$ are in the same equivalence class in R_1 (Figure 1). P_1 is the probability distribution over the elements in an equivalence class in R_1. We assume that P_1 is uniform since the player does not have any information about her opponent's hand.

There are two propositional symbols in this example, w_1 and w_2, to show whether player 1 or player 2 wins the hand respectively. In one state at most one of them can be true. A tie is presented with both w_1 and w_2 being false. The value of w_1 and w_2 (which is part of V) is predetermined for each state based on the game rules.

After modeling Holdem with a probabilistic knowledge structure, we would like to be able to answer if a formula (query) is true in our model or not. In the following section we focus on different reasoning methods to answer a query.

4 Reasoning Algorithms

4.1 Exact Reasoning: Top-Down & Bottom-Up

In this section we provide reasoning algorithms that evaluate queries (formulas) about a given state. In Holdem, $K_1(K_2(w_2) < 1/4) > 2/3$ is an example of a query. If Alice and Bob are player 1 and player 2 respectively, this query refers to Alice's beliefs about Bob's beliefs. She believes that with probability greater than $2/3$ Bob believes

FUNCTION ToDo(state s, query q)
"value" associates with each state, a truth assignment to the primitive propositions. "$P_i(s'|s)$" is the probability of accessing state s' after being in state s.
1. if $q = \top$ then return *true*
2. elseif $q = \bot$ then return *false*
3. elseif q is a propositional letter then return value(s, q)
4. elseif $q = \neg x$ then return [$\neg$ ToDo(s, x)]
5. elseif $q = x \wedge y$ then
   return [ToDo(s, x) $\wedge$ ToDo(s, y)]
6. elseif $q = (K_i(x)\ \alpha\ r)$ then
   (a) $m \leftarrow 0$
   (b) for all states s' accessible from s
   if ToDo(s', x) then $m \leftarrow m + P_i(s'|s)$
   (c) return $(m\ \alpha\ r)$

Figure 2: Top-Down (ToDo) reasoning algorithm.

that with probability $\geq 3/4$ he is losing. Therefore, if Bob is raising, Alice knows that it is likely he is bluffing.

Computing the value of a query is straightforward in theory given a probabilistic modal structure (definition 2.2). Given the structure answering a query with no modal function can be done in O(length of query). The expensive part of the computation of a query is the part with modal function. Assume that the number of states accessible from a state is n. The number of states that should be visited to calculate the value of a query is multiplied by n for each nested modal function (e.g., $O(n^m)$ states should be visited when we have m nested modal functions).

We can represent a query with an expression tree (Kozen 1997). In an expression tree, the root is the query itself and the leaves are propositional primitives. Function Top-Down (ToDo) of Figure 2 starts from the root of the query's expression tree and recursively computes the value of its subformulas on a given state. The running time of the function grows exponentially with the number of nested modal functions.

Theorem 4.1 *Let q be the query whose value on state s we want to compute and let $|q|$ be the length of the query. Let m be the number of nested modal functions and let n be the maximum number of states accessible from any state. Function ToDo calculates the truth value of the query on state s in $O(|q| \times n^m)$ time.*

In function ToDo subformulas may be computed multiple times. We can overcome this inefficiency, if we take a bottom-up approach. There, we start from the bottom of the query's expression tree and avoid computing a subformula multiple times. Function Knowledge-Bottom-Up (KBU) of Figure 3 computes the value of a query about a probabilistic knowledge structure while it avoids recalculation.

We first compute the value of an innermost modal function (with no nested modal function in it) for all equivalence classes, and associate the results with all the states in their respective equivalence classes. For computational purposes, we implement this (line 2(a) in function KBU) with a temporary propositional symbol in all states. Then, we replace the modal function in the query with the temporary proposition. We continue this operation until all modal functions in the query are computed. In function KBU, we visit each

FUNCTION KBU(state s, query q)
"P_i" associates with each state the probability of being in that state given that we are in its equivalence class. "val" recursively calculates the value of a formula with no modal function.
1. $m \leftarrow 1$
2. for all $(K_i(x)\ \alpha\ r)$ in q where x has no modal function
   (a) for all $e \in$ equivalency classes of relation i
   - $t_m \leftarrow [\sum_{s' \in e, \text{val}(s',x)=true} P_i(s')]\ \alpha\ r$
   - add t_m to $\forall s' \in e$
   (b) $q \leftarrow$ replace $(K_i(x)\ \alpha\ r)$ in q with t_m
   (c) $m \leftarrow m + 1$
3. return val(s, q)

FUNCTION GBU(state s, query q)
1. $m \leftarrow 1$
2. for all $(K_i(x)\ \alpha\ r)$ in q where x has no modal function
   (a) add $t_{m,true} \leftarrow 0$, $t_{m,false} \leftarrow 0$, t_m to $\forall s_1 \in$ set of states
   (b) for all $s_2 \in$ set of states
   - for all s_1 that s_2 is accessible from s_1
   $t_{m,\text{val}(s_2,x)} \leftarrow t_{m,\text{val}(s_2,x)} + P_i(s_2|s_1)$
   (c) for all $s_1 \in$ set of states
   - $t_m \leftarrow [t_{m,true}/(t_{m,true} + t_{m,false})]\ \alpha\ r$
   (d) $q \leftarrow$ replace $(K_i(x)\ \alpha\ r)$ in q with t_m
   (e) $m \leftarrow m + 1$
3. return val(s, q)

Figure 3: Bottom-Up reasoning algorithms.

state once for each modal function.

Theorem 4.2 *Let q be the query whose value on state s we want to compute. Let $|S|$ be the number of states in the connected component graph of state s. Function KBU calculates the value of the query q on state s in $O(|q| \times |S|)$ time.*

Compared to the runtime of function ToDo which is exponential in the number of nested modal functions, the runtime of KBU is linear in the length of the query (i.e. total number of modal functions). As a drawback, KBU is linear in the total number of states. When the number of nested modal functions is small, function ToDo is faster than KBU. To choose the best method for our application, we compare n^m with $|S|$ and decide whether to use ToDo or KBU.

The bottom-up approach can be extended to general probabilistic modal logic. In probabilistic modal logic we have a directed graph of states. Each edge in this graph corresponds to an accessibility relation of a modal function. For example, in a two player Holdem game an edge is related to either K_1 or K_2. To extend the bottom-up algorithm to general modal logic, we need to know which states are accessible from a particular state. For example to calculate $K(x)$ on state s_1 we need to add the conditional probability $P(s_2|s_1)$ of all states s_2 in which formula x is true.

To calculate $K(x)$ on all the states in a probabilistic modal logic structure, we first calculate x on each state. Suppose that x is true (false) in state s_2. For each s_1 that has a directed edge corresponding to modal function K toward s_2, we add $P(s_2|s_1)$ to the temporary variable that we use for the probability of the states accessible from s_1 in which x is true (false). The value of $K(x)$ easily can be calculated after performing this operation on each state. The General-

FUNCTION BEST(state s, query q)
1. if $q = K_i(x)$ then
   (a) $m \leftarrow 0$
   (b) for all states s' accessible from s
       if ToDo(s', x) then $m \leftarrow m + P_i(s'|s)$
   (c) return m
2. else return ToDo(s, q)

Figure 4: The algorithm for computing the *best* probability.

Bottom-Up (GBU) algorithm for reasoning in general modal logic structures is shown in Figure 3.

Theorem 4.3 *Let q be the query whose truth value on state s we want to compute. Let $|S|$ be the number of states in the connected component of state s. Function GBU calculates the value of query q on state s in $O(|q| \times |S|^2)$ time.*

The proof intuition is that each edge is visited once for each modal function. The number of edges in a dense graph is quadratic in the number of nodes.

Although the bottom-up algorithms are faster than top-down method when the number of nested modal functions grows, it is still costly when the number of states is large.

Function ToDo can be slightly improved if we avoid unnecessary calculation. We can avoid calculating a formula for the same set of states repeatedly by keeping track of what we have already calculated. This is specially useful when the formula has a modal function. Suppose that we have two states: s_1 and s_2. They are indistinguishable to both player 1 and player 2 (they are in the same equivalence class). Now, assume that we want to answer the query $K_1(K_2(K_1(x) < a) < b) < c$ on state s_1. By our method, function $K_2(K_1(x) < a) < b$ should be calculated on states s_1 and s_2 which are both accessible from s_1. We first calculate $K_2(K_1(x) < a) < b$ for one state (s_1) and then the other (s_2). We need to calculate $K_1(x) < a$ for both s_1 and s_2, to answer the query $K_2(K_1(x) < a) < b$ on state s_1 and also later to answer the same query on state s_2. If we keep track of the value of $K_1(x) < a$ on s_1 and s_2 we can reuse it without any computation.

Another unnecessary computation happens in answering $X \wedge Y$. If X is false, the value of Y does not matter. As we showed before, the calculation of formulas with more nested modal functions is more expensive than the others. Therefore, in the case of $X \wedge Y$ it is more efficient to first calculate the formula with fewer nested modal functions. It may avoid the computation of the other one.

Best probability. So far, our algorithms assumed that we are interested in finding the truth value of a query about a given a Kripke structure. For example, if player 1 (Alice) believes that $K_1(w_1) < 1/3$ is true, she probably will not call the other player's raise. However, sometimes it is more important for Alice to know the *best* value r such that $K_1(w_1) < r$ holds. Based on this number she has a better estimate of her chance of winning.

Figure 4 extends our earlier algorithms and provides an algorithm that computes the best probability about a query (e.g. minimum r such that $K(x) < r$ is true). It can use either KBU or GBU instead of ToDo. The input to this algorithm is of the form $K(x)$. The algorithm returns the probability of x from the perspective of modal function K.

FUNCTION ARea(state s, query q)
"M" is a constant (number of samples). "Pa" associates with each node in a graph the list of its parents.
1. define $T(q, s)$ based on definition 4.4
2. $N \leftarrow$ reverse all the edges in $T(q, s)$
3. for each node l in N with operator $\wedge$
   - for each v, v_1, v_2 in $\{0, 1\}$
     $CPT_N(l = v|Pa(l) = (v_1, v_2)) \leftarrow (v = v_1 \wedge v_2)$
4. for each node l in N with operator $\neg$
   - for each v, v' in $\{0, 1\}$
     $CPT_N(l = v|Pa(l) = (v')) \leftarrow (v = \neg v')$
5. for each node l in N with operator $K_i(x) < r$
   - for each $v, v_1, \ldots, v_M$ in $\{0, 1\}$
     $CPT_N(l = v|Pa(l) = (v_1, \ldots, v_M)) \leftarrow$ Formula (1)
6. return inference(N, $Pr(q_s = 1|l_1 = v_1, \ldots, l_r = v_r)$)
   % $l_1, \ldots, l_r$ are all the nodes in N with no parent (we have their value $v_1, \ldots, v_r$ in the tree)

Figure 5: Approximate Reasoning (ARea) algorithm.

4.2 Sampling Subgraphs

In this section we provide a reasoning method that uses sampling to answer queries on probabilistic knowledge structures. As we showed in the previous section, the running time of the bottom-up reasoning methods depend on the number of states. In our poker example, the number of states reachable from an initial state given all five boardcards is equal to $\binom{47}{2}\binom{45}{2}$. Therefore, the bottom-up approach is tractable. However, in a two player game with hundreds of cards when each player has tens of cards, it is not practical to go through all the states multiple times to answer a query.

All the states accessible from a state should be visited to evaluate a formula with a modal function. If the number of states accessible from a state is too large, evaluating $K(x)$ on that state would be expensive. To avoid this expensive computation, we sample the states and calculate x on a smaller set. Then we calculate the probability of $(K(x)\ \alpha\ r)$ given the sampled data. Since the probability distribution is continuous, in this section we do not allow equality in the query (i.e., $K(x) = r$).

The function Approximate Reasoning (ARea) presented in Figure 5 returns the probability of a query given the sampled data. This probability serves as an estimate for the truth value of the query. We use sampling to compute $K(x) < r$. We calculate the number of sampled states in which x is true and use it to estimate $K(x)$. The size of the sampled set is fixed throughout the program. This method is much faster than the exact methods when the number of states explodes.

The value of a formula on a sampled state is either true or false. So, the number of sampled states in which formula x is true has a binomial distribution with parameter $K(x)$. To calculate $K(x) < 1/3$ on state s, we sample M states from the probability distribution $\frac{P(s')}{\sum_{s'' Rs} P(s'')}$(we label them $s_1, \ldots, s_M$). We compute $Pr(K(x) < 1/3|x_{s_1}, \ldots, x_{s_M})$where x_{s_i} represents the value

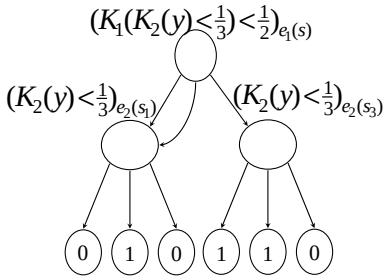

Figure 6: $T(K_1(K_2(y) < 1/3) < 1/2, e_1(s))$.

of x at state s_i. We define the *sampled expression tree* of a formula as follows. From this point on we use 1 and 0 instead of true and false.

Definition 4.4 *Let q be a query that we want to evaluate on state s. Let $e_i(s)$ be the equivalence class of state s in R_i (where R_i is the accessibility relation corresponding to K_i). The* sampled expression tree $T(q, s)$ *of query q on state s is:*

- *If $q = \top$ then $T(q, s)$ is a node whose value is 1.*
- *If $q = \bot$ then $T(q, s)$ is a node whose value is 0.*
- *If $q = r$ and r is a propositional letter then $T(q, s)$ is a node whose value is equal to the value of r in state s.*
- *If $q = \neg x$ then $T(q, s)$ is a node whose operator is $\neg$ and it has one child $T(x, s)$.*
- *If $q = x \wedge y$ then $T(q, s)$ is a node whose operator is $\wedge$ and it has two children $T(x, s)$ and $T(y, s)$.*
- *If $q = (K_i(x) < r)$ then instead of $T(q, s)$ we define $T(q, e_i(s))$ since the values of q on all the states in $e_i(s)$ are the same. $T(q, e_i(s))$ is a node whose operator is q and it has M children $T(x, s_1), \ldots, T(x, s_M)$ when $s_1, \ldots, s_M$ are the M samples in $e_i(s)$.*

We assign a label to each node such that the label of the root of $T(q, s)$ is q_s. Whenever we define $T(K_i(x) < r, e_i(s))$ for the first time, we sample M states from the probability distribution $\frac{P(s')}{\sum_{s'' R_i s} P(s'')}$.

The sampled expression tree for query $K_1(K_2(y) < 1/3) < 1/2$ is shown in figure 6 where $M = 3$. Here, s_1 and s_2 are in the same equivalence class therefore $(K_2(y) < 1/3)_{e_2(s_1)}$ and $(K_2(y) < 1/3)_{e_2(s_2)}$ are the same.

In algorithm ARea, we transform $T(q, s)$ to a *Bayesian network* (Pearl 1998). First, we reverse the direction of all the edges. Then, we define the Conditional Probability Table (CPT) for each node (which is the probability distribution of the node given its parents). Defining the CPT for a $\wedge$ node is straight-forward. $Pr((x \wedge y)_s = 1 | x_s = 1, y_s = 1) = 1$ where x_s and y_s are the two parents of the node, and the probability of $(x \wedge y)_s = 1$ given any other combination of the parents' values is 0. This is the natural way to define $\wedge$. For a $\neg$ node $Pr((\neg x)_s = 1 | x_s = 0) = 1$ and $Pr((\neg x)_s = 0 | x_s = 1) = 1$ when x_s is the parent of the node.

We define the CPT for node $K_i(x) < r$ as follows:

$$Pr((K_i(x) < r)_{e_i(s)} = 1 | x_{s_1} = v_1, \ldots, x_{s_M} = v_M) = \frac{\int_0^r f(K_i(x) = u) u^{M_t} (1-u)^{M-M_t} du}{\int_0^1 f(K_i(x) = u) u^{M_t} (1-u)^{M-M_t} du} \quad (1)$$

when $x_{s_1}, \ldots, x_{s_M}$ are the M parents of the node, and $M_t = \sum_{i=1}^{M} v_i$. Here, $K_i(x)$ is a random variable with probability density function (pdf) $f(K_i(x) = u)$. To estimate the query given the sampled data, the pdf of $K_i(x)$ should be known. For simplicity, we assume that $f(K_i(x) = u) = 1$, that is, the prior probability distribution is uniform.

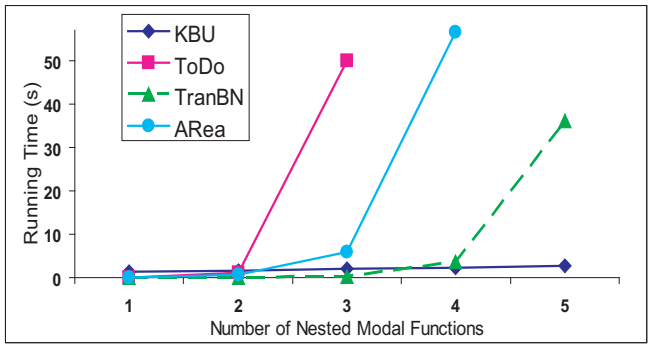

Figure 7: Running time diagram. Number of states $= \binom{47}{2}\binom{45}{2}$

In this Bayesian network two or more parents of a node may be the same (e.g., Bayesian network of Figure 6). In that case, their value should also be equal. This does not affect the CPT except that some of the combination of parents' values never happen. The algorithm returns the probability of the query given the values of the leaves of $T(q, s)$.

Theorem 4.5 *Let M be the number of samples at each stage. Let q be the query that we want to evaluate on state s. The algorithm ARea returns the probability of $q = 1$ given the values of the leaves of $T(q, s)$ in time exponential in the tree-width of $T(q, s)$ (Kschischang, Frey, & Loeliger 2001; Huang & Darwiche 1996).*

After reducing the problem to inference in Bayesian networks, different approximation methods can be used to calculate the probability (Jordan *et al.* 1999; Yedidia, Freeman, & Weiss 2004).

5 Experimental Results

In this section we compare the running time of different exact and approximate reasoning algorithms. We run all the methods on a probabilistic knowledge structure of our Holdem example. As shown in Figure 7, the running time of ToDo grows exponentially with the number of nested modal functions while KBU grows linearly. In typical real-world situations the number of nested modal functions in queries is small. In Holdem, most of the times the number of nested modal functions is ≤ 3. The largest degree of modal nesting that a player may care about is the belief of his opponent about his belief. Therefore, if we know in advance that the degree of nesting is small, we may prefer to use ToDo. Notice that ToDo is only faster when the number of nested modal functions is small. In applications in which the degree of nesting is high, KBU is the better option.

We can take advantage of both algorithms by writing a function that uses ToDo for queries with small number of nested modal functions (m) and KBU otherwise. Based on theorem 4.1 and 4.2, comparing n^m with $|S|$ gives us a good intuition about which algorithm to use.

Figure 7 also compares the running time of our approximation method with the exact ones. It shows that when the

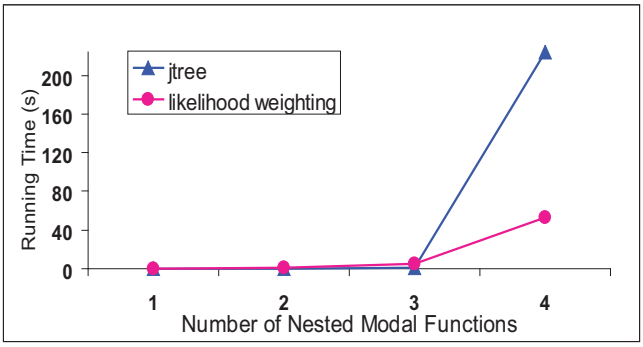

Figure 8: InfBN running time ($M = 10$).

number of states is not very large, using the approximation does not make sense. The approximation should be used only when we have infinitely many or very many states.

Our approximation algorithm (ARea) consists of two parts: transforming the query to a Bayesian Network (using sampling), and performing inference on that network. We refer to them as TranBN and InfBN, respectively. As shown in Figure 7, the running time of TranBN is $O(|q| \times M^m)$ when M is the number of samples used for calculating modal functions, m is the number of nested modal functions in the query, and $|q|$ is the size of the query. The size of the Bayesian network is also $O(|q| \times M^m)$. The running time of InfBN depends on the algorithm that we use for inference. We used Bayes Net Toolbox (BNT) for Matlab written by Kevin Murphy [1] in our experiments.

There are many inference algorithms in BNT, each of which make different tradeoffs between speed, accuracy, complexity and generality. In Figure 7, we use a Monte Carlo sampling inference algorithm called likelihood_weighting_inf_engine. It applies importance sampling and can handle any node type. We also try the junction tree engine (jtree_inf_engine), which is the mother of all exact inference algorithms. Figure 8 compares the running time of these two inference engines on our examples. After transforming the query to a Bayesian network, any inference algorithm with any implementation can be used (as InfBN).

ARea returns an approximation to the value of the query. It calculates the probability of the query being true given the sampled data. For some queries this probability is high ($> .9$) even for a small set of samples. In those cases ARea returns a good approximation to the value of the query. However, when a modal function is compared to a number close to its actual value, the sampling may not work well (e.g., the query is $K(x) < r$ and $K(x)$ is close to r). In those cases the probability of the query given sampled data (the output of ARea) shows our lack of confidence in the value of the query.

6 Conclusion & Future Work

We provided a syntax and semantics for reasoning with probabilistic modal logic. We suggested exact and approximate algorithms for evaluating queries over a probabilistic Kripke structure. We showed that exact methods are not tractable when we have a very large structure. However, our approximate algorithm returns an estimate (probability of the query given a sampled model) efficiently.

An important direction for future work is investigating belief update in our language. Given an action and an observation how to change the structure of the world. Also, given a set of probabilistic modal logic formulas KB, it is important to answer whether a query holds or not. In this paper, we do not have any inference $KB \models \varphi$ beyond a straightforward model enumeration or the special case of (Fagin & Halpern 1988). There are many other questions that can be raised with this paper. "How to learn a probabilistic modal logic structure?", "How to perform planning?", etc.

[1] Available from: http://bnt.sourceforge.net/

7 Acknowledgements

We wish to acknowledge support from Army W9132T-06-P-0068 (CERL Champaign), Army DACA42-01-D-0004-0014 (CERL Champaign), and NSF IIS 05-46663 (US National Science Foundation).

References

Aumann, R. J. 1986. Reasoning about knowledge in economics. In *TARK*.

Blei, D.; Ng, A.; and Jordan, M. 2003. Latent dirichlet allocation.

Fagin, R., and Halpern, J. Y. 1988. Reasoning about knowledge and probability. In *TARK*.

Fagin, R.; Halpern, J.; Moses, Y.; and Vardi, M. 1995. *Reasoning about knowledge*. MIT Press.

Fitting, M. 1993. Basic modal logic. In *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 1: Logical Foundations*.

Friedman, N., and Halpern, J. Y. 1994. A knowledge-based framework for belief change, part i: Foundations. In *TARK*.

Halpern, J. Y. 1987. Using reasoning about knowledge to analyze distributed systems. In *Annual Review of Computer Science*.

Heifetz, A., and Mongin, P. 1998. The modal logic of probability. In *TARK*.

Herzig, A. 2003. Modal probability, belief, and actions. *Fundam. Inf.* 57(2-4):323–344.

Huang, C., and Darwiche, A. 1996. Inference in belief networks: A procedural guide. *IJAR* 15(3):225–263.

Jordan, M. I.; Ghahramani, Z.; Jaakkola, T.; and Saul, L. K. 1999. An introduction to variational methods for graphical models. *Machine Learning* 37(2):183–233.

Kozen, D. C. 1997. *Automata and Computability*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Kschischang, F.; Frey, B.; and Loeliger, H. 2001. Factor graphs and the sum-product algorithm. *IEEE Trans. Inform. Theory*.

Milch, B., and Koller, D. 2000. Probabilistic models for agents' beliefs and decisions.

Nie, J.-Y. 1992. Towards a probabilistic modal logic for semantic-based information retrieval. In *SIGIR*. ACM Press.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. Morgan Kaufmann.

Pearl, J. 1998. Bayesian networks. Technical Report 980002.

Yedidia, J.; Freeman, W.; and Weiss, Y. 2004. Constructing free energy approximations and generalized belief propagation algorithms.