First-Order Open-Universe POMDPs: Formulation and Algorithms

Siddharth Srivastava and **Xiang Cheng** and **Stuart Russell**
EECS Department
University of California, Berkeley

Avi Pfeffer
Charles River Analytics
Cambridge, MA

Abstract

Interest in relational and first-order languages for probability models has grown rapidly in recent years, and with it the possibility of extending such languages to handle decision processes—both fully and partially observable. We examine the problem of extending a first-order, open-universe language to describe POMDPs and identify non-trivial representational issues in describing an agent's capability for observation and action—issues that were avoided in previous work only by making strong and restrictive assumptions. We present a method for representing actions and observations that respects formal specifications of the sensors and actuators available to an agent, and show how to handle cases—such as seeing an object and picking it up—that could not previously be represented. Finally, we argue that in many cases open-universe POMDPs require belief-state policies rather than automata policies. We present an algorithm and experimental results for evaluating such policies for open-unverse POMDPs.

1. Introduction

Relational and first-order languages for probability models (as well as their close relatives the probabilistic programming languages) constitute an important development for AI in general and for machine learning in particular (Getoor and Taskar 2007). The availability of such expressive languages should make it possible to write not just complex probability models but also complex decision models—the foundation for rational behavior in autonomous agents. To achieve this goal, probabilistic languages can be extended with information about actions, observations, and rewards. As we show, however, such extensions raise significant difficulties. We argue that as machine learning and probabilistic AI researchers grapple with these more expressive representations, it will be necessary to use techniques from logical AI in conjunction with probabilistic techniques to produce representations that are both expressive enough to model the real world and mean exactly what is intended. In this paper, we use such techniques to develop a representation of first-order decision models. We also describe methods for evaluating expressive belief-state policies for such models.

Decision models for agents operating in partially observable non-deterministic environments are expressed as partially-observable MDPs (POMDPs). The observation and action models for the agent play a key role in a POMDP and naturally, determine its optimal solution policies. When we move to the relational or first-order setting, we have to contend with the fact that observations, rather than being values of variables, become sentences describing properties of objects using names for those objects. It turns out to be quite tricky to say in a formal way what a given sensor can or cannot supply in the way of observation sentences; expressing the agent's capability for action is also tricky, especially when the agent can act on objects whose existence has been determined only through its sensors. Consider the following example:

The sensors of an airport security system include passport scanners at check-in kiosks, boarding pass scanners, X-ray scanners, etc. A person passing through the airport generates observations from each of these scanners. Thus, the passport scanner at location A may generate observations of the form *IDName*$(p_{A,1})$ = *"Bond"*, *IDNumber*$(p_{A,1}) = 174666007$, *HeightOnID*$(p_{A,1}) = 185cm$, ...; a boarding-pass scanner at B may generate a sequence of the form *Destination*$(p_{B,7})$ = *"Paris"*, *IDNumber*$(p_{B,7}) = 174666007$, and finally, an X-ray scanner at C may generate observations of the form *MeasuredHeight*$(p_{C,32}) = 171cm$, *MeasuredHeight*$(p_{C,33}) = 183cm$.

In these observation streams, the symbols $p_{A,i}, p_{B,j}$ and $p_{C,k}$ are place-holder identifiers (essentially Skolem constants or "gensyms" in Lisp terminology). Although each use of a given symbol necessarily corresponds to the same individual, different symbols may or may not correspond to different individuals; thus, it is possible that $p_{A,1}$ and $p_{C,32}$ refer to the same person, while it is also possible that $p_{A,1}$ and $p_{B,7}$ refer to different people even though they are carrying a passport with the same ID number.

Such a scenario can be modeled probabilistically by a first-order, open-universe language such as BLOG (Milch et al. 2005), which enables reasoning about identity uncertainty. To make *decisions*—such as searching or arresting a given individual—we need a POMDP with rewards, actions, and an observation model. Informally, we might say, "Everyone in the security line will get scanned":

$$\forall\, x \;\; \textit{InLine(x)} \rightarrow \textit{Scanned(x)}$$

and "For everyone who gets scanned, we will observe a measured height":

$$\forall x \quad \textit{Scanned(x)} \rightarrow Observable(\textit{MeasuredHeight(x)}). \quad (1)$$

So far, so good. Now, suppose we know, "Bond and his fiancee are in the security line." While it is true, in a sense, that we will get a measured height for Bond's fiancee, it is *not* true that the X-ray scanner will tell us:

$$\textit{MeasuredHeight(Fiancee(Bond))} = 171cm.$$

Technically, the problem arises because we are trying to substitute *Fiancee(Bond)* for x in the universally quantified sentence (1), but one occurrence of x is inside *Observable*$(\cdot)$, which is a *modal operator*. Practically, the problem arises because the sensor doesn't know who Bond's fiancee is. The same issue can arise on the action side: telling the security guard to "Arrest Bond's fiancee" doesn't work if the guard doesn't know who Bond's fiancee is.

These issues stem from two fundamental, related problems that this paper focuses on: overstating what can be sensed and passing commands with arguments that don't refer to anything meaningful for actuators.

In addition to presenting a formal language for defining OUPOMDPs, we explain how policies may be represented and we describe algorithms for evaluating such policies. We focus in particular on policies that depend directly on the belief state—in contrast with many papers in the literature that focus on policies expressed as observation–action trees or automata—because the observation space of an OUPOMDP may be unbounded. In the airport example, the agent's belief state includes states with any number of persons in the queue, and any person being at the scanner. In such a belief state, the set of possible actions and observations—even if sensor and actuator specifications are captured accurately—is infinite. On the other hand, a first-order language can be used to specify effective high-level policies of the form: "if the probability of the person at the scanner being a terrorist is above a certain threshold, arrest that person". We present a novel approach that ensures the conformance of such policies to the available sensors and actuators during their evaluation.

We begin in §2. by describing how POMDPs may be defined as directed graphical models by extending dynamic Bayesian networks (DBNs). We focus in particular on the *observation model*. In order to extend these ideas to first-order open-universe POMDP specifications, we being with system-independent models of sensors and actuators. We avoid the pitfalls of standard solutions by defining a meta-predicate representing the ability to observe something and formal semantics capturing the set of allowed decisions and the observations that may be expected when the agent is in a given state (§4.). We show that the resulting framework models sensor and actuator specifications accurately. We also develop a novel algorithm for evaluating OUPOMDP policies (§5.) and present several results from an implementation (§6.). Finally, we discuss the ways in which previous attempts to define first-order POMDPs have used strongly restricted languages to avoid these problems altogether. These restrictions do not allow an agent to walk into a room, see something, and pick it up (§7.).

2. POMDPs

A POMDP defines a decision-theoretic planning problem for an agent. At every step, the agent executes an action, then the environment enters a new state, then the agent receives an observation and a reward.

Definition 1. *A POMDP is defined as $\langle Q, A, O, T, \Omega, R, \gamma \rangle$, where Q, A, O are finite sets of state, action and observation symbols; $T(Q_{t+1} = q' \mid Q_t = q, A_t = a)$ defines the transition model, i.e., the probability of reaching state q' if action a is applied in state q; $\Omega(O_{t+1} = o \mid Q_{t+1} = q', A_t = a)$ defines the observation model, i.e., the probability of receiving an observation o when state a' is reached via action a; and $R : Q \times A \rightarrow \mathbb{R}$ defines the reward that the agent receives on applying a given action at a given state. The rewards obtained by the agent are aggregated via a discounted sum, $\sum_{i=1\ldots\infty} \gamma^i \cdot r(i)$ where $r(i)$ is the reward obtained at timestep i and $\gamma \leq 1$.*

A belief-state in a POMDP is a probability distribution over the set of states Q. Since the agent may only know a belief-state rather than the true state at each timestep, POMDP solutions have to map belief-states to actions. In most representations, such mappings take the form of a function that maps observation histories to actions (Kaelbling, Littman, and Cassandra 1998). The optimal policy maximizes the expected value of the total discounted reward obtained.

A DBN (Dean and Kanazawa 1989) describes a factored, homogeneous, order-1 Markov process as a "two-slice" Bayesian network showing how variables at time $t + 1$ depend on variables at t. At each time step, any subset of variables may be observed. To represent POMDPs, Russell and Norvig 1995 assume a fixed set of always-observable evidence nodes and define *dynamic decision networks* (DDNs) as extensions of DBNs with action and reward node:

- An *action variable* A_t whose values are the possible actions at time t. For now, assume this set of actions is fixed. The POMDP's transition model is represented through the conditional dependence of variables at $t + 1$ on the value of A_t and other variables at t.
- A reward variable R_t, which depends deterministically on A_t and other variables at time t.

A general model for POMDPs however needs to specify which nodes will be evidence nodes at a given timestep—different specifications correspond to different POMDPs. In order to avoid a not-missing-at-random (Little and Rubin 2002) scenario when the set of evidence nodes is not fixed, we can define for each variable X that may be observed, a second Boolean variable $ObsX$ that captures whether or not X is observed [1]. This factors out dependencies for observability from dependencies for the variable values. Thus, in order to define POMDPs we can define DDNs consisting

[1] An alternative would be to say that "null" values are observed when a variable is not observable. However, this approach has distinct disadavantages as it requires (a) unnecessarily large parent sets for evidence variables capturing when null values may be obtained, and (b) additional mechanisms for handling dependencies of child nodes of variables that may get a null value.

of the following nodes in addition to the action and reward nodes:

- A set of Boolean *observability variables* $ObsX_t$, one for each ordinary variable X_t. Each observability variable is necessarily observed at each time step, and $ObsX_t$ is true iff X_t is observed. Observability variables may have as parents other observability variables, ordinary variables, or the preceding action variable.[2] The DBN with X and $ObsX$ variables defines Ω.

In this model, an always-observable X_t has an $ObsX_t$ with a deterministic prior set to 1.

In order to define first-order OUPOMDPs, we need to extend first-order probability models in a manner analogous to the DDN extension of DBNs. However, as we will show below, the analogous extension leads to conflicts with what may be known to the agent during decision making, and results in the models of sensors and actuators with unintentionally broad capabilities as seen in the introduction. We begin with a brief summarization of the terminology of first-order logic and open-universe probability models.

3. First-Order Probability Models

First-Order Vocabularies Given a set of types $\mathcal{T} = \{\tau_1, \ldots \tau_k\}$, we define a first-order vocabulary as a set of function symbols with their type signatures. Constant symbols are represented as zero-ary function symbols and predicates as Boolean functions. Given a first-order vocabulary, a structure is defined as a tuple $\langle \mathcal{U}, \mathcal{I} \rangle$ where the universe $\mathcal{U} = \langle \mathcal{U}_1, \ldots \mathcal{U}_k \rangle$ and each $\mathcal{U}_i$ is a set of elements of type $\tau_i \in \mathcal{T}$. The interpretation $\mathcal{I}$ has, for each function symbol in the vocabulary, a function of the corresponding type signature over $\mathcal{U}_1, \ldots, \mathcal{U}_k$. The set of types includes the type *Timestep*, whose elements are the natural numbers. Functions whose last argument is of type timestep are called *fluents*. We represent possible worlds using structures. The values of all static functions and fluents with timestep fixed at t denote the state of a possible world at timestep t. We represent actions by defining the value of a fluent at timestep $t+1$ in terms of non-fluents and fluents at timestep t. The reward function can be expressed as a fluent in a similar manner (cf. Reiter's (2001) successor-state axioms). In order to define probabilistic action effects under full observability, these ideas can be extended to include a probability distribution for all possible action effects (Sanner and Boutilier 2009).

Open-Universe Probability Models in BLOG Our approach can be applied to formal languages for generative, open-universe probability models (OUPMs). BLOG (Milch et al. 2005; Milch 2006) is one such language. We refer the reader to the cited references for details on this system, and discuss briefly the components relevant to this paper. A BLOG model consists of two types of statements: (1) number statements, which specify conditional probability distributions (cpds) for the number of objects of each type in the universe of a structure; and (2) dependency statements,

[2] Observability variables capture the full range of possibilities in the spectrum from missing-completely-at-random (MCAR) to not-missing-at-random (NMAR) data (Little and Rubin 2002).

```
1 Type Urn, Ball;
2 origin Urn Source(Ball);
3 #Urn ~ Poisson(5);
4 #Ball(Source = u) {
5  if Large(u) then ~ Poisson(10)
6  else ~ Poisson(2)};
7 random Boolean Large(Urn u)
8   ~ Bernoulli(0.5);
```

Figure 1: A BLOG model illustrating number statements.

which specify cpds for the values of functions applied on the elements of the universe.

Each type can have multiple number statements and each number statement can take other objects as arguments. Fig. 1 shows a simple example of a BLOG model with two types, *Urn* and *Ball*. Intuitively, the model expresses a distribution over possible worlds consisting of varying numbers of urns with varying numbers of balls in each urn. The number of urns follows a Poisson(5) distribution (line 3). The number of balls in an urn depends on whether or not the urn is *Large*. *Origin* functions map the object being generated to the arguments that were used in the number statement that was responsible for generating it. In Fig. 1, *Source* maps a ball to the urn it belongs to. The number of balls in an urn follows a Poisson(10) distribution if the urn is *Large*, and a Poisson(2) distribution otherwise (lines 4-6). Finally, whether or not an urn is *Large* follows a Bernoulli(0.5) distribution (lines 7 & 8).

BLOG can express dependencies capturing successor-state axioms in a straightforward manner. For example, a *sendToScanner(x, t)* action may result in the person x going to the scanner. Let *followedInstructionCPD*, *leftScannerCPD* and *defaultScannerCPD* denote respectively the probability distribution that a person follows instructions, that s/he has left the scanner and that s/he is already at the scanner. The following dependency captures the effect of this action on the predicate *atScanner*.

```
random Boolean atScanner(Person x, Timestep t+1) {
  if applied_sendToScanner(x,t)
    then ~ followedInstructionCPD()
  else if atScanner(x, t)
    then ~leftScannerCPD()
  else ~defaultAtScannerCPD()};
```

These formulations do not address which terms can be used in action arguments or observations. A natural generalization of the $obsX$ idea discussed in §2. is to write rules of the form: $observable(\psi(\bar{x}))$ $\{$*if* $\varphi(\bar{x})$ *then* $\sim cpd_1\}$, where φ and ψ are arbitrary FOL formulas. WLOG, ψ and φ can be considered to be predicates defined using FOL formulas with variables in $\bar{x}$ as free variables. The interpretation of this formula would be "$\psi(\bar{x})$ is observed with probability given by cpd_1 when $\varphi(\bar{x})$ holds". Problems with this framework that were discussed in the introduction are consequences of the axiom of universal instantiation in first-order logic. This axiom states that if $\forall x\ \alpha(x)$ is true, then for any ground term t, $\alpha(t/x)$ (the version of $\alpha(x)$ where all free occurrences of x are replaced by t) must also be true. The example in the introduction was a result of substituting *Fiancee(Bond)* for x in Eq. 1. In modal contexts such as observability, this axiom has to be restricted (Levesque and

Lakemeyer 2000): we can only substitute x with terms that are, in some sense, "known" to the agent. In the following sections we utilize this concept to develop the semantics for observation and decision statements.

4. OUPOMDP Models in DTBLOG

In this section we present the key components of decision-theoretic BLOG (DTBLOG), which adds to the BLOG language decision variables for representing actions and meta-predicates for representing observability. The declarative semantics for these extensions clarify the sensor and actuator specifications they define. DTBLOG models can also be constructed with such specifications as inputs to the modeling process. The procedural semantics for these extensions define the DTBLOG engine. These semantics will be used to compute the observations possible in a belief state as well as to generate the possible effects of executing decisions on a given belief state.

4.1 Sensor/Actuator Specifications

We first define representation-independent mathematical specifications of sensors and actuators. These specifications describe the types of vectors returned by sensors and accepted by actuators.

Definition 2. *A* sensor specification S *is a tuple* $\langle \bar{T}_S, \tau_S \rangle$ *where* $\bar{T}_S$ *is a tuple of types and* τ_S *is a type.*

$\bar{T}_S$ defines the type-vector of observation values that S produces and τ_S defines the type of new symbols that it may generate. E.g., an X-ray scanner can be specified as $scanner = \langle \langle$*PersonRef, Real*$\rangle, \langle$*PersonRef*$\rangle \rangle$. Such a scanner can generate new symbols of type *PersonRef* and returns observation tuples of the type $\langle$*PersonRef*, *Real*$\rangle$.

Definition 3. *An* actuator specification A *is a tuple of types* $\bar{T}_A$ *denoting the types of its arguments.*

An actuated camera may be able to take a picture given an orientation and focusing distance: *TakePhoto* $= \langle$*Orientation*, *Real*$\rangle$.

Any OUPOMDP definition has to specify a set of available actions, state transition system, observation function and the reward function. Independent of these components, we can use sensor and actuator specifications to define desirable OUPOMDP definitions as those which enforce the use of unambiguous terms of the correct types in the possible decisions and observations.

Definition 4. *An OUPOMDP is well-defined wrt a set of sensor specifications* $\mathcal{S}$ *and actuator specifications* $\mathcal{A}$ *iff every decision for an actuator* $A \in \mathcal{A}$ *and every evidence statement attributed to a sensor* $S \in \mathcal{S}$ *has as its arguments terms of types prescribed by* A *and* S *respectively. Further, in the agent's belief state where evidence is obtained or a decision made, (a) each term used as an argument in an evidence statement for* S *must have a unique evaluation in terms of elements of the universe or symbols generated by* S *and (b) each term used as an argument in a decision for* A *must have a unique interpretation as an element of the universe.*

4.2 Sensors in DTBLOG

As noted above, sensors provide two kinds of inputs to the agent: symbol observations represent the symbols that they generate, and relational observations capture sensed properties. In the following, the declarative semantics define a set of sensor specifications in DTBLOG. The procedural semantics of the DTBLOG engine ensure that when it generates an observation, it respects the sensor specifications as well as constraints on the agent's available knowledge.

Declarative Semantics Given a sensor $S = \langle \bar{T}_s, \tau_s \rangle$, we model τ_s as a type whose extension is the set of symbols generated by that sensor. We specify such a sensor S using the following components in DTBLOG:

- A predicate V_S with arguments $\bar{T}_S, t$, representing the tuples returned by the sensor.
- The statement `ObservableType(`τ_s`);` denoting that symbols of type τ_s are returned by the sensor. Number statements for τ_S constitute a generative model for the elements generated by S. Each number statement for a sensor symbol of type τ includes a origin function τ_time which maps the symbol to the time when it was generated.
- A dependency for *Observable*$(V_S(\bar{T}_S,$ *Timestep*$))$, denoting the conditions when the sensor S is likely to generate a relational observation.

A DTBLOG model for the X-ray scanner can be represented as:

```
ObservableType(PersonRef);
#PersonRef(Src = p, PersonRef_Time=t) {
   if AtScanner(t)=p then ~Bernoulli(0.5)
   else = 0};
random Bool Observable(V_scanner(p, h, t)) {
  if AtScanner(source(p), t) then ~Bernoulli(0.5)
  else = false};
```

For ease in representation, we also allow syntax for capturing sensors that return function values. For instance, it may be convenient to represent the scanner as a sensor that provides values of the measured height, captured by the function $MeasuredHt_{scanner}(p, t)$. In this case, the relational observability statements would provide dependencies for *Observable*$(MeasuredHt_{scanner}(p, t))$.

Procedural Semantics Intuitively, if the environment is in state q, and *Observable*$(\varphi(\bar{x}))$ (or *ObservableType*(τ)) is true in s, then the value of $\varphi(\bar{x})$ (or all symbols of type τ) must be obtained as evidence in the state q. The procedural semantics for DTBLOG implement this intuition, while ensuring that x is only substituted with terms known to the agent.

Symbol Observations All elements generated via a sensor's symbol observability statement are assigned unique names and provided to the agent as an evidence statement. This is achieved by compiling a statement of the form *ObservableType*(τ) into:

```
random Int Number_τ(t)=#{τ x:  τ_time(x) ==
t};
Observable(Number_τ(t))=true;
```

This compilation uses the procedural semantics for observability of relations, discussed below. If the model in-

cludes the statement *ObservableType*(τ), the DTBLOG engine generates evidence statements for the symbols of type τ at each timestep. For a state where where $k(t) = Number\_\tau(t)$, the DTBLOG engine provides an evidence statement of the form:

```
obs {τ c:τ_time(c) == t} = {c_1, ..., c_k(t)};
```

The semantics of BLOG ensure that $c_1, \ldots, c_{k(t)}$ are interpreted as distinct objects.

Relational Observations For every true *Observable*($\varphi(\bar{x})$) atom in a state, the DTBLOG engine creates an observation statement where all arguments are "evaluated". E.g.,

```
obs MeasuredHt_scanner(pref17, 1) = 171;
```

Each argument in such evidence statements is evaluated-out and can only be (a) a predefined symbol with a fixed interpretation, (b) a symbol generated by the same sensor or (c) the application of any number of deterministic functions on (a) and (b). In particular, evidence statements generated by the DTBLOG engine only use arguments of the form (a) and (b). Interactive sessions where a user provides evidence to the engine may provide observations that have partially evaluated terms of the form (c).

This formulation allows models to express accurately the terms that are observable and can be used in observations. Returning to the informal example described in the introduction, under this formulation the DTBLOG engine will not generate an observation of the form *MeasuredHeight(Fiancee(Bond)) = 150cm* if the value of *Fiancee(Bond)* has not been observed, even when *MeasuredHeight()* is observable for all persons.

4.3 Actuators in DTBLOG

Decision variables are declared using the keyword `decision`.

Declarative Semantics An actuator specification $A = \bar{T}_A$ is specified in DTBLOG as:

```
decision apply_a(T̄_A, Timestep);
```

For example the actuated camera can be specified as:

```
decision apply_TakePhoto(Orientation,
Distance,
Timestep);
```

Procedural Semantics A user can provide values for decision variables either interactively or through a policy specification. Without any further restrictions, this would lead to unintended situations where the user provides a decision of the form:

```
apply_TakePhoto(Orientation(Loc(Src(pref17),
t)),
DistanceTo(Loc(Src(pref17), t)), t)=true;
```

even when *Loc(Src(pref17), t)* has not been observed. Such a decision would not only be meaningless to the actuator, it can lead to "fake" solutions, e.g. if the desired effect was to take a picture of the person who generated *pref17* at the scanner, but has since moved away.

The DTBLOG engine evaluates terms in a decision assertion only if (a) their values have been observed, (b) their values are fixed in the model, or (c) they are constructed using deterministic functions applied on terms of the form (a) and (b). If a decision includes as its arguments terms that were not observed, state update subroutines update the state without applying the decision.

A full DTBLOG model for the airport domain example is presented in the appendix.

Let $M(\mathcal{S}, \mathcal{A})$ be a DTBLOG model defined using the sets $\mathcal{S}$ and $\mathcal{A}$ of sensors and actuators respectively. Let $\mathcal{V}_M$ be the first-order vocabulary used in M. Then, M defines an OUPOMDP $\langle Q, A, O, T, \Omega, R\rangle$ where the set of states Q is the set of states corresponding to the possible worlds of vocabulary V. A is the set of all instantiated decision functions corresponding to $\mathcal{A}$ that are allowed in some state $q \in Q$ and O is the set of all instantiated functions corresponding to sensor specifications in $\mathcal{S}$ that are allowed in some state $q \in Q$. The transition function T and observation function Ω are defined by the probabilistic dependency statements in M.

Note that our formulation does not place any constraints on the successor-state axioms or the dependencies for values of observations. Since a BLOG model must include dependencies for every declared function, these components have to be defined in any DTBLOG model whose vocabulary includes the decision variables and observation relations corresponding to sensors. The following result follows from the procedural semantics above and shows that this formulation corresponds to a modal logic of observed information rather than of the complete knowledge possessed by the agent.

Lemma 1. *The procedural semantics of DTBLOG ensure that (a) arguments in the evidence statements generated by the DTBLOG engine for a sensor S only use symbols that are generated by S or are predefined and thus have unique interpretations in the agent's belief-state (b) values of terms used as arguments in decisions have been observed or are predefined and thus have unique interpretations in the agent's belief-state.*

In other words, the terms that the agent may expect in an observation or that it uses in a decision in a given belief-state have unique values in all states with non-zero probability under that belief-state. Lemma 1 leads to the main result of this section.

Theorem 1. *Led $\mathcal{M}$ be a DTBLOG model defined using sensor and actuator specifications $\mathcal{S}$ and $\mathcal{A}$ respectively. If $\mathcal{M}$ satisfies BLOG's requirements for well-defined probabilistic models then it constitutes a well-defined OUPOMDP model wrt $\mathcal{S}$ and $\mathcal{A}$.*

Actions on Sensor-Generated Symbols For representational convenience, we also allow the use of sensor-generated symbols in actions that can be compiled down to primitive actions respecting the semantics defined above. Consider a situation where the scanner reports the estimated location of the person generating a person reference (the function *Location* maps person references to locations) in addition to their measured heights. We then define a camera action that takes a snapshot given a *PersonRef*. In the following example, the functions *RelativeOrientation* and *CamDistance* map positions to the orientation and distance relative to the camera, respectively.

```
apply_TakePhotoPRef(p_ref, t) :=
apply_TakePhoto(RelativeOrientation(Location(p_ref)),
   CamDistance(Location(p_ref)), t)
```

Every instance of *apply_TakePhotoPRef()* is compiled out into the primitive action *apply_TakePhoto()*. This allows the agent to act on the objects detected through its sensors. A generative model may specify when the effect of an action on a person reference is likely to have the desired effect. The notion of such high-level actions can be developed further. For instance, one could define an action that, given a *PersonRef*, determines the maximum likelihood estimate for the position of the person who generated that reference and takes a picture of that location. Such actions would have to be specified outside the DTBLOG model since they need to execute queries on the model itself to construct their arguments. Probabilistic effects of such actions however have to be defined in the model to be consistent with their external definitions. Automatically constructing the effect descriptions of such actions is left for future work.

Belief States and Transitions in DTBLOG DTBLOG represents belief-states using collections of sampled, possible states. The initial belief state is generated using BLOG's existing sampling subroutines to sample possible worlds corresponding to the state at timestep 0 specified in the DTBLOG model. The application of a decision updates each possible world to the next timestep using the stated dependencies (aka probabilistic successor-state axioms).

As a notable consequence of the semantics and belief state representation used in DBTLOG, when the belief state is updated wrt to a decision, the DTBLOG engine generates the set of observations corresponding to each possible updated state.

5. OUPOMDP Policies

We consider two types of policy representations in this paper. Finite-state controller (FSC) policies map sequences of observations to actions. FSC policies can take the form of tree-structured contingent policies or cyclic controllers (Hansen 1998) and are widely used in the POMDP literature. However, in the case of OUPOMDPs such policies can be very difficult to construct. In the airport domain, for example, searching in the space of FSC policies amounts to considering all possible sequences of ID and biometric measurements stemming from arbitrary numbers of persons. Instead, we focus on developing a general policy evaluation framework that applies on FSC policies as well as a general class of belief-state query (BSQ) policies. BSQ policies map the results of first-order queries on open-universe belief states to actions. A BSQ policy has the form the form *if* $Pr(\varphi_1(x)) \in I_1$ *then do* $a_1(x)$*; else if* $Pr(\varphi_2(x)) \in I_2$ *then do* $a_2(x)$ ..., where $\varphi_k(x)$ are first-order formulas and $I_k \subseteq [0, 1]$ are intervals. The variable x is implicitly existentially quantified and bound to a value that satisfies φ_k, if any. High-level domain specific BSQ policies are often easier to specify for OUPOMDPs. For instance one could simply say *if the probability of Src(p_ref) being a terrorist and leaving the area soon is more than* θ*, then detain the person at estimated location of Src(p_ref)*. On the other hand, it would be very hard to express a similar policy using an FSC policy. If

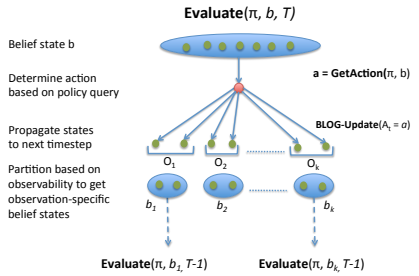

Figure 2: Illustration of one iteration of Alg. 1

Algorithm 1: EvalPolicyBFS

Input: DTBLOG model $\mathcal{M}$, initial belief state b_0, policy π, horizon h

Step $\leftarrow 0$; $b \leftarrow b_0$;
if *Step* $< h$ **then**
  a $\leftarrow$ GetAction(π, b);
  b'$\leftarrow$ BLOG_Update(b, a);
  $\{b'_1, \ldots, b'_k\} \leftarrow$ ObservabilityPartition(b');
  return Average($EvalPolicyBFS(\mathcal{M}, b'_1, h - 1), \ldots, EvalPolicyBFS(\mathcal{M}, b'_k, h - 1)$);
return AverageValue(b');

a BSQ policy evaluation algorithm is available, partial BSQ policies of this form can be effectively refined into concrete BSQ policies by identifying appropriate values for parameters like θ.

5.1 Approximate Policy Evaluation

Policy evaluation algorithms for OUPOMDPs need to address two main concerns. Algorithms that assume the feasibility of considering all possible observations as the result of action application on a belief state are infeasible, because OUPOMDP belief states may capture possible worlds with unknown and unbounded numbers of objects. As a result, the number of possible observations resulting from an application of an action on a belief state can be infinite. A secondary concern arises when evaluating BSQ policies: the process of obtaining the action to be applied using a BSQ policy requires an estimation of the agent's current belief state. Computationally this is much more expensive than determining the action to be applied under an FSC policy, which only needs to look-up the action corresponding to an observation sequence. The need for state estimation while evaluating BSQ policies precludes the use of existing sampling-based algorithms that do not keep track of the agent's belief state (e.g., PEGASUS (Ng and Jordan 2000)).

The policy evaluation algorithm presented below works for both representations by keeping a sampled estimate of the agent's current belief state. Before describing our main algorithm, we describe the necessary elements using a version (Alg. 1) that conforms to existing ideas of policy evaluation for POMDPs. This algorithm correctly evaluates OUPOMDPs, but is unlikely to perform well in practice. We then introduce an algorithm based on particle filtering to address the first algorithm's limitations and prove that it converges to the correct value estimate. We assume that the

number of possibly observable random variables is finite but unbounded in each possible world. Note that the number of possible observations from a belief state can still be infinite, since there is no upper bound on the number of possible worlds in a belief state.

At each timestep, the EvalPolicyBFS algorithm (Alg. 1; Fig. 2) essentially applies an action on a sampled representation of the possible belief states and samples the set of observations to branch over. The initial belief state b_0 is captured by the DTBLOG model $\mathcal{M}$. In every successive timestep, the algorithm maintains collections of sampled possible worlds representing the current possible belief states. In order to update a belief state into the next timestep, it uses an external policy specification π on a belief state b to determine the action $\pi(b)$ to be applied This determination incorporates the knowledge considerations described in §4.3, which may lead to a no-op action being applied if $\pi(b)$ does not satisfy the required conditions. We then update all the sampled worlds representing b to the next timestep. This is done by setting the decision variable corresponding to a to be true in each possible world and then propagating each world to the next timestep by sampling the values of random variables corresponding to the next timestep. This provides the updated belief state b' (line 4). Since observable functions are treated like any other functions in our formulation, this update also instantiates their values, as well as the values of the observable() meta-predicates.

As discussed earlier, the set of functions for which observable() is true in each world determines the observations that are produced for that member of the belief state. The algorithm partitions the possible worlds according to observations received in them. This results in a set of sampled belief states $\{b'_1, \ldots, b'_k\}$, one corresponding to each unique combination of observations among the possible worlds representing b' (line 5). The execution then recurs, calling EvalPolicyBFS on each of these belief states with the target horizon decremented. In the terminal case of this recursion, the algorithm returns the average of the value function for all possible worlds in the updated belief state b'.

Alg. 1 uses sampling to address the problem of branching over infinitely many possible observations. However, since it has to start with a finite set of possible worlds representing the initial belief state and partitions them in each timestep, its variance increases rapidly with the timestep due to a shrinking number of sampled worlds in each successive belief state.

Our main algorithm (Alg. 2) overcomes this difficulty by using a particle filter to follow different sequences of possible observations (Fig. 3). This is done by selecting at each timestep, at random, one of the possible worlds in the belief state corresponding to that timestep, and using its observations as evidence. Thus, each iteration of the while loop (line 3) corresponds to a particle filter that maintains exactly one belief state at each timestep and follows a sequence of sampled observations.

Theorem 2. *The result of EvalPolicyDFS($\mathcal{M}, b_0, \pi, h$) converges to the true expected value of π after h timesteps starting with the belief state b_0 wrt the OUPOMDP $\mathcal{M}$ in the limit of infinitely many samples and infinitely many particles per belief state.*

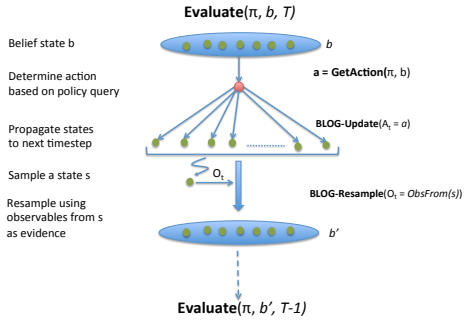

Figure 3: Illustration of one iteration of Alg. 2

Algorithm 2: EvalPolicyDFS

Input: DTBLOG model $\mathcal{M}$, initial belief state b_0, policy π, horizon h

```
1 repeat
2     Step ← 0;
3     while Step < h do
4         a ← GetAction(π, b);
5         b'← BLOG_Update(b, a);
6         w ← SampleWorld(b'); o ← ObservationsFrom(w);
7         b_{Step+1} ← BLOG_ReSample(b', o ); Step++;
8     Add AverageValue(b_{Step}) to SetOfValueSamples;
  until SampleLimit reached;
9 return Average(SetOfValueSamples);
```

Proof. By convergence results for particle filters, a single run of EvalPolicyDFS converges to the expected value of the policy while following a particular sequence of observations. When run with different random seeds, we get a set of sampled values, each corresponding to a particular sequence of possible observations when following π. Convergence to the true expected value is guaranteed because at each timestep, the probability of selecting a particular observation approaches the likelihood of that observation in the limit of infinitely many samples in the initial belief state. □ □

Alg. 2 has several properties that make it desirable for practical use. The main operations of updating particles can be carried out in parallel, and each observation sequence can be evaluated in parallel. It computes the expected value of a path in time linear in the number of timesteps and the number of particles.

6. Empirical Results

We describe the results of our experiments with an implementation of Alg. 2 on two domains.

Tigers This problem is an open-universe version of the popular Tiger POMDP (Kaelbling, Littman, and Cassandra 1998). The agent is in a circular room with 10 doors. There are an *unknown* number of tigers behind the doors, who may move from a door to its neighbor at each timestep. Multiple tigers may be behind a door. The objective is to open a door without a tiger behind it and enter it.

The agent has two actions, a *listen(Timestep t)* action that allows it to make inaccurate observations about the sounds made by tigers at timestep t, and an *enter(Door d, Timestep*

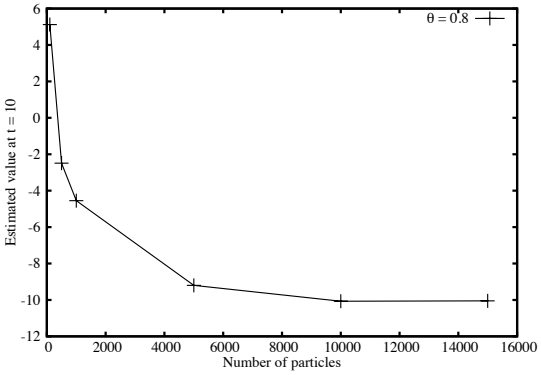

Figure 4: Estimated values with increasing particle count ($\theta = 0.8$).

t) action which it can use to open a door and enter. When listen is applied, the agent obtains a sound from each tiger with probability 0.5.

```
random Bool observable(Sound) = true;
#Sound(Source = m, Time_Sound = t) {
 if apply_listen(t-1)
   then ~ Bernoulli(0.5)
 else = 0};
```

The listen action also gives a noisy estimate of the doors from which sounds came. If a sound is made by a tiger at door d, the probability of observing that a sound was made at d is 0.75, and that of observing that a sound was made at each of the doors $(d + 1 \bmod 10)$ and $(d - 1 \bmod 10)$ is 0.25. At each timestep, each tiger stays behind its current door with probability 0.4 and moves to each of the neighboring doors with probability 0.3. The agent receives a reward r_{safe} for entering a door without tiger, r_{listen} for listening, and r_{danger} for entering a door with a tiger behind it.

The unknown number of tigers, their movement, and the data association problem of matching sounds to tigers make it hard to represent observation-sequence based policies in this domain. However, we can write a belief-state query policy $\pi(\theta)$ of the form:
if Pr(no tiger behind d_1 at t) $> \theta$, $enter(d_1, t)$
else if Pr(no tiger behind d_2 at t) $> \theta$, $enter(d_2, t)$
...
else $listen(t)$

We used Alg. 2 to estimate the value of this BSQ policy for different values of θ. Once we obtain a set of randomly generated sequences of observations and decisions following $\pi(\theta)$ for a particular value of θ, we can analytically compute the expected value for any setting of $r = (r_{safe}, r_{listen}, r_{danger})$ and γ since they don't change the decision and observation functions. We used γ as 1, and the horizon as 10 in all the experiments. As a baseline, we used the policy which listens until it finds a door without sounds and enters it.

Figs. 4 and 5 summarize the results. Each point is an average of 3 runs, with each run using 100 observation paths. The standard deviation across runs was less than 1.6 for runs with at least $10,000$ particles. Fig. 4 shows that the estimated value converges as we increase the number of particles for $\pi(0.8)$. Fig. 5 shows the estimated value of $\pi(\theta)$

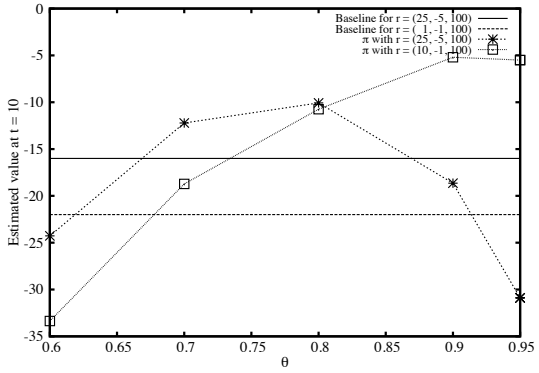

Figure 5: Estimated values for $\pi(\theta)$.

for $\theta = 0.6, 0.7, 0.8, 0.9, 0.95$ for $r = (25, -5, -100)$ and $r = (10, -1, -100)$. The results show that for high-reward situations with a high penalty for listening, $\theta = 0.8$ performs best. On the other hand if the potential reward and the penalty for listening are low, $\theta \geq 0.9$ performs best, as it is better to be sure of safety before opening a door.

These results show that Alg. 2 can effectively be used to optimize a parametric family of BSQ policies by searching for parameters that yield high expected values.

Blind Monopoly This problem models a version of the game Monopoly with 20 squares and two players: the agent and an opponent. The players take turns rolling two die to move along the board, and can purchase a property they land on, provided it hasn't already been bought. If a player lands on a property belonging to another, s/he must pay rent to the owner. If a player owns all properties of a color, the rents of those properties are doubled. The game is "blind", in the sense that a player cannot observe the opponent's holdings or positions. Its only observation about the opponent is whether or not it receives or pays rent at a timestep. The objective is to have more cash than the opponent at $t = 100$. The initial rent was 35; properties cost 60; and there were 5 colors with three squares each in a pattern similar to the actual board game (some properties cannot be bought). The opponent's policy was to purchase the first available property it landed on. It then purchased any other property of the same color in an effort to complete the set. It also bought other properties with probability 0.5. In this version of monopoly, it is very difficult to formulate an FSC policy that allows the agent to complete a set: such a policy has to map the past rent observations, to decisions that should depend on whether or not it is still possible to complete the agent's current set or to block the opponent by purchasing from a different color.

However, BSQ policies can be designed easily. We used a policy where the agent purchases the first property it lands on, but makes more purchases than the opponent based on its belief about the opponent's holdings. In all purchases other than for completing its set (the first rule below) it executes a purchase only if it has sufficient funds to pay at least one more rent. It's BSQ policy is to apply the first rule whose premise holds:

(Complete Set) If the agent owns a property of the current

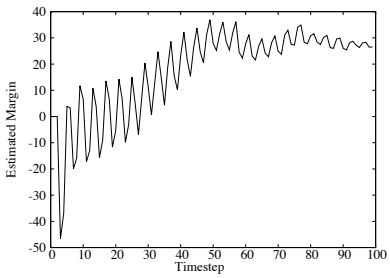

Figure 6: Win margin while following a BSQ policy in blind monopoly.

color, and Pr(opponent *not* owning one of this color) $>$ θ_1, buy

(Block) If the agent doesn't own a property of the current color, and Pr(opponent owning one of this color) $> \theta_2$, buy

(Randomize) Buy with probability 0.5.

Fig. 6 shows the agent wining at t=100. It plots the evolution of the expected value of the agent's capital minus the opponent's capital for $\theta_1 = 0.75, \theta_2 = 0.5$. We used 200 observation paths with 5000 particles.

7. Related Work and Conclusions

To the best of our knowledge, Moore (1985) presented the first comprehensive FOL formulation of actions that did not make the unique names assumption and allowed terms in the language to be partially observable, in a non-probabilistic framework. In Moore's formulation actions could be executed by an agent only if they were "known" to it. This notion of epistemic feasibility of an action was also used in later work (Morgenstern 1987; Davis 1994; 2005). These approaches used a significantly larger axiomatization to address the problem of syntactically proving and communicating facts about knowledge. However, this line of work cannot be used in open-universe probabilistic languages due to the requirement of reifying possible worlds and terms as objects in a universe. It also does not address the problem of expressing observability and action availability while conforming to a given agent specification.

Our formulation of action effects uses update rules similar to successor state axioms proposed by Reiter (2001). However, usually employed assumptions like having a "closed initial database" in that line of work preclude the possibility of expressing identity uncertainty: distinct terms like *Mary* and *Fiancee(Bond)* can never represent the same object. Sanner et al. (2010) use this framework for first-order POMDPs and make the additional assumption that all non-fluent terms are fully observable. They suggest a *same-as*(t_1, t_2) predicate for representing identity uncertainty between fluent terms. However, it is not clear how this predicate can be used in conjunction with their unique names axioms for actions, which assert that instances of an action applied on distinct terms must be distinct. Wang et al. (2010) present a relational representation for POMDPs while making the unique names and closed world assumptions: in their framework, action arguments have to be in a known 1-1 mapping with actual objects in the universe, implying that every object has, a priori, a unique name in the language; observations directly report properties of such objects. Thus, representational constraints in prior work on first-order models for POMDPs disallow the expression of key aspects of open-universe semantics (e.g. is Mary=Fiancee(Bond)?). In addition, in contrast to existing approaches, our formulation allows an agent to plan and act upon objects *discovered* through its sensors. Such problems are commonly encountered by agents in the real world. Unfortunately, as we showed, standard "extensions" of existing frameworks to handle these problems lead to grossly inaccurate definitions of an agent's capabilities.

Recent algorithms for sampling based search for POMDP policies (Silver and Veness 2010; Guez, Silver, and Dayan 2012) also use a particle filter to carry out belief updates while searching for history-based policies. Our utilization of particle filters for evaluating BSQ policies differs on two main aspects. First, existing approaches rely upon simulations of single state trajectories to obtain an estimate of the expected value while following a history based policy. As noted earlier, such trajectories are not sufficient for evaluating (or even simulating the execution of) BSQ policies. Second, we utilize the generative model in a weighted particle filter rather than the unweighted filter used in the aforementioned approaches, which treat the POMDP as a black box model.

Various authors have considered POMDP solutions that directly map belief states, rather than observation histories, to actions. BSQ policies offer a compact, expressive form for such policies. In recent work Kaelbling et al. (2013) proposed an approach for solving partially observable problems by carrying out regression-based planning over belief states. In their approach, action specifications are designed to include preconditions in the form of belief-state fluents. These fluents can be defined to capture probabilistic queries, which in turn can be used to indicate when an action is likely to succeed or be helpful in achieving a goal. However, the solution approach requires action-specific regression functions over the probabilities of such queries.

Our approach builds on several of the approaches discussed above to solve the unaddressed problem of accurately specifying agent models under first-order open-universe semantics, and of evaluating BSQ policies for such models. Our framework is unique in allowing accurate representations of planning problems encountered by autonomous agents in the real-world (as first-order OUPOMDPs). This is done by drawing upon modal logic semantics. Further, our experiments showed that optimizing the parameters of a BSQ policy family can be a viable approach for solving OUPOMDPs.

Acknowledgments

This research was supported in part by DARPA DSO grant FA8650-11-1-7153, DARPA contract W31P4Q-11-C-0083 and the NSF under grant number IIS-0904672.

References

Davis, E. 1994. Knowledge preconditions for plans. *J. Log. Comput.* 4(5):721–766.

Davis, E. 2005. Knowledge and communication: A first-order theory. *Artif. Intell.* 166(1-2):81–139.

Dean, T., and Kanazawa, K. 1989. A model for reasoning about persistence and causation. *Comput. Intell.* 5(3):142–150.

Getoor, L., and Taskar, B. 2007. *Introduction to Statistical Relational Learning*. The MIT Press.

Guez, A.; Silver, D.; and Dayan, P. 2012. Efficient bayes-adaptive reinforcement learning using sample-based search. In *NIPS*, 1034–1042.

Hansen, E. A. 1998. Solving POMDPs by searching in policy space. In *In Proc. of UAI*, 211–219.

Kaelbling, L. P., and Lozano-Pérez, T. 2013. Integrated task and motion planning in belief space. *I. J. Robotic Res.* 32(9-10):1194–1227.

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

Levesque, H. J., and Lakemeyer, G. 2000. *The logic of knowledge bases*. MIT Press.

Little, R. J. A., and Rubin, D. B. 2002. *Statistical analysis with missing data (second edition)*. Wiley.

Milch, B.; Marthi, B.; Russell, S. J.; Sontag, D.; Ong, D. L.; and Kolobov, A. 2005. BLOG: Probabilistic models with unknown objects. In *Proc. of IJCAI*, 1352–1359.

Milch, B. C. 2006. *Probabilistic models with unknown objects*. Ph.D. Dissertation, University of California at Berkeley, Berkeley, CA, USA.

Moore, R. C. 1985. *A Formal Theory of Knowledge and Action*. Ablex. 319–358.

Morgenstern, L. 1987. Knowledge preconditions for actions and plans. In *Proc. of IJCAI*, 867–874.

Ng, A. Y., and Jordan, M. I. 2000. PEGASUS: A policy search method for large MDPs and POMDPs. In *Proc. of UAI*.

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. Massachusetts, MA: The MIT Press.

Russell, S. J., and Norvig, P. 1995. *Artificial Intelligence - A Modern Approach: The Intelligent Agent Book*. Prentice Hall.

Sanner, S., and Boutilier, C. 2009. Practical solution techniques for first-order MDPs. *Artif. Intell.* 173(5-6):748–788.

Sanner, S., and Kersting, K. 2010. Symbolic dynamic programming for first-order POMDPs. In *Proc. of AAAI*.

Silver, D., and Veness, J. 2010. Monte-carlo planning in large pomdps. In *Advances in Neural Information Processing Systems*, 2164–2172.

Wang, C., and Khardon, R. 2010. Relational partially observable MDPs. In *Proc. of AAAI*.

A DTBLOG Model: Airport

This appendix provides a DTBLOG model for the airport domain. Type declarations are omitted.

```
#Person ~ Poisson[10]; LocKnownDuration=4;
#PersonRef(Src = p, PersonRef_Time=t) {
   if AtScanner(t)=p ~Bernoulli(0.5)
   else = 0};
observable(PersonRef);
observable(MeasuredHt(p_ref, t))=(AtScanner(t) == Src(p_ref));
decision apply_TakePhoto(Orientation o, Distance d, Timestep t);
Ht(prsn) ~ Normal(160, 30);
MeasuredHt(p_ref, t) ~ Normal[Ht(Src(p_ref), t), 5];
PictureTaken(Person p, Timestep t){
 if t',t>0 & exists PersonRef p_ref Src(p_ref)==p
   & PersonRef_Time(p_ref) == t'
   & apply_TakePhotoPRef(p_ref, t-1)
   & t-1-t'< LocKnownDuration then == True
 else if t>0 then = PictureTaken(p, t-1)
 else = False}
//Entrance model
AtScanner(t) ~ UniformChoice({Person prsn:
  !Entered(prsn, t)});
Entered(prsn, t){
  if t>0 & (AtScanner(t-1)=prsn) then = true
  elseif t>0 then = Entered(prsn, t-1)
  elseif t=0 then = false};
TrueLoc(Person p, t) ~ MovementModel(loc(p), t-1);
Location(p_ref, t) {if PersonRef_Time(p_ref)<Horizon+1 &
  & t<=Time_P_Ref(p_ref)+LocKnownDuration
    then ~ MovementModel(TrueLoc(Src(p_ref), t))
    else = null};
```