First-Order Open-Universe POMDPs

Siddharth Srivastava and **Stuart Russell** and **Paul Ruan** and **Xiang Cheng**
Computer Science Division
University of California
Berkeley, CA 94720

Abstract

Open-universe probability models, representable by a variety of probabilistic programming languages (PPLs), handle uncertainty over the existence and identity of objects—forms of uncertainty occurring in many real-world situations. We examine the problem of extending a declarative PPL to define decision problems (specifically, 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 semantic definitions that lead to POMDP specifications provably consistent with the sensor and actuator capabilities of the agent. We also describe a variant of point-based value iteration for solving open-universe POMDPs. Thus, we handle cases—such as seeing a new object and picking it up—that could not previously be represented or solved.

1 INTRODUCTION

The development of probabilistic programming languages or PPLs (Koller, McAllester, and Pfeffer, 1997; Milch et al., 2005; Goodman et al., 2008) has greatly expanded the expressive power of formal representations for probability models. In particular, PPLs express *open-universe probability models*, or OUPMs, which allow uncertainty over the existence and identity of objects. OUPMs are a natural fit to tasks such as vision, natural language understanding, surveillance, and security, where the set of relevant objects is not known in advance and the observations (pixels, strings, radar blips, login names) do not uniquely identify the entities in question.

It is natural, therefore, to consider whether the same benefits can be obtained for decision models, thereby providing a broader foundation for rational agents. A general decision model—a partially observable Markov decision process or POMDP—includes a specification of what the agent can do and what it will perceive in any given state, as well as the reward functions.

For less expressive languages such as Bayesian networks and closed-universe first-order languages, the extension from probability models to decision models is relatively straightforward. For open-universe models, however, there are significant difficulties in defining POMDP representations that are both expressive enough to model the real world and mean exactly what is intended. When sensors supply sentences about named objects and actuators receive commands to act on named objects, problems arise if the formal names have uncertain referents. 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 documents with the same ID number.

The problems in modeling such a scenario are not limited to probabilistic models and can be illustrated using first-order

logic alone. For the observation model, we might want to say that, "Everyone in the security line will get scanned":

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

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

$$\forall x \;\; \mathit{Scanned}(x) \rightarrow \mathit{Observable}(\mathit{MeasuredHeight}(x)). \tag{1}$$

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:

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

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*; thus, we have a failure of referential transparency—perhaps not surprising as we are trying to model what the agent will come to know. 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. Thus, any proposed formal approach has to provide solutions to three fundamental problems: how to incorporate detected objects in an agent's belief while allowing identity uncertainty; how to accurately state what can be sensed, and how to ensure that arguments in commands refer to something meaningful for actuators.

In fact, communication between the physical layer and the formal model requires a restricted vocabulary whose terms are guaranteed to be meaningful. This is because in a probability model, an observation has to be *true* to be conditioned on. The problem is that an OUPOMDP framework needs to ensure this even when the observations use uncertain references. We formally define the terms that will be meaningful, and therefore suitable for use in communication with the physical layer, using the concept of *rigid designators* from modal logic. This notion facilitates a framework where observations are true statements, without being restrictive. Our solution is analogous to modeling uncertain observations of a property as certain observations about a noisy version of that property.

We begin in §2.1 by showing, as a "warm-up exercise," how POMDPs may be defined on a substrate of dynamic Bayesian networks (DBNs). §2.2 provides additional background on first-order, open-universe probability models. §3 describes our proposed semantics for a decision-theoretic extension of the BLOG language. We show that the resulting framework models sensor and actuator specifications accurately. §4 describes a variant of the point-based value iteration (PBVI) algorithm designed to handle open-universe POMDPs, and §5 describes experiments in a simple domain that illustrate the ability of the formalism and algorithm to handle problems that previous formalisms (§6) cannot express and previous algorithms cannot solve. It has been possible for many years to program a robot to walk into a room, see something, and pick it up; now, it is possible for the robot to do this without leaving the formal framework of rational decision making.

2 BACKGROUND

2.1 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* $\langle X, U, O, T, \Omega, R, \gamma \rangle$*, where* X, U, O *are finite sets of states, actions and observations;* $T(X_{t+1} = x' \mid X_t = x, U_t = u)$ *defines the transition model, i.e., the probability of reaching state* x' *if action* u *is applied in state* x*;* $\Omega(O_{t+1} = o \mid X_{t+1} = x', U_t = u)$ *defines the observation model, i.e., the probability of receiving an observation* o *when state* x' *is reached via action* u*; and* $R : X \times U \rightarrow \mathbb{R}$ *defines the reward that the agent receives on applying a given action at a given state.*

The agent's belief state at any timestep is the probability distribution over X at that timestep, given the initial belief and the history of executed actions and obtained observations. POMDP solutions typically map observation histories to actions (Kaelbling, Littman, and Cassandra, 1998). We will refer to such solutions as *observation-history* policies. On the other hand, *belief-state policies* map the agent's belief states to actions. Both solution representations are equally expressive because the belief state constitutes a sufficient statistic for the agent's history and the initial belief (Astrom, 1965). An optimal POMDP policy policy maximizes the expected value of the aggregate reward $\sum_{i=1\ldots\infty} \gamma^i \cdot r(i)$ where $r(i)$ is the reward obtained at timestep i and $\gamma \leq 1$.

A 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 timestep, 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) by adding to each timestep of a DBN the following variables:

- An *action variable* U_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 U_t and other variables at t.
- A reward variable R_t, which depends deterministi-

cally on U_t and other variables at time t.

This definition of POMDPs by DDNs is not, however, completely general, since it does not allow for the observability of a variable to depend on the state. Clearly, different observability conditions correspond to different POMDPs. To handle state-dependent observability, one can define for each variable X_i that may be observed, a second Boolean variable $ObsX_i$ that captures whether or not X_i 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 of the following nodes in addition to the action and reward nodes:

- A Boolean *observability variable* $ObsX_i$ for each ordinary variable X_i. Each observability variable is necessarily observed at each time step, and $ObsX_i$ is true iff X_i is observed. Observability variables may have as parents other observability variables, ordinary variables, or the preceding action variable. The DBN with X_i and $ObsX_i$ variables defines Ω.

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

In order to define first-order OUPOMDPs, we need to extend first-order OUPMs 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.

2.2 OPEN-UNIVERSE PROBABILITY MODELS

First-Order Vocabularies Given a set of *types* $\mathcal{T} = \{\tau_1, \ldots \tau_k\}$, a first-order *vocabulary* is a set of function symbols with their type signatures. Constant symbols are viewed as zero-ary function symbols and predicates as Boolean functions. A *possible world* 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*. A state is defined by the values of all static functions and fluents at a particular timestep in a possible world.

[1]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). 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.

```
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       ~ Categorical{true->0.5, false->0.5};
```

Figure 1: A BLOG model illustrating number statements.

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, 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*. This 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, the probability of an urn being *Large* is 0.5 (lines 7 & 8).

Evidence statements in BLOG provide first-order sentences in the model's vocabulary as observations; e.g.,

obs (exists Ball b exists Urn u Source(b)==u & Large(u)) = true

states that there exists a large urn that has a ball.

A *set evidence* statement provides a concise syntax for naming all the distinct objects satisfying a certain predication; e.g.,

obs {Ball b: exists Urn u Source(b)==u & !Large(u)}={b1, b2}

states that the set of balls in small urns consists of exactly two balls, referred to by the constant symbols $b1$ and $b2$.

We denote the execution of an action $u(x)$ at a timestep t using the fluent *apply_u*$(\bar{x}, t)$. The effects of an action are represented by defining the value of all fluents affected by the action at timestep $t + 1$ in terms of the fluents and non-

fluents at timestep t. The reward function can be expressed as a fluent in a similar manner. This notation is similar to successor-state axioms in situation calculus (Reiter, 2001). For example, a *sendToScanner*(p, t) action may result in the person p 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. We express the effect of this action on the predicate *atScanner* using the fluent *apply_sendToScanner*(x, t) as follows:

```
atScanner(Person p, Timestep t+1) {
    if apply_sendToScanner(p,t)
        then ~ followedInstructionCPD()
    else if atScanner(p, t) then ~ leftScannerCPD()
    else ~ defaultAtScannerCPD()};
```

Representation theorems for BLOG ensure that every well-founded specification (that does not create cyclic dependency statements or infinite ancestor sets) corresponds to a unique probability distribution over all possible worlds.

3 DECISION-THEORETIC BLOG

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.

3.1 SOLUTION APPROACH

A natural generalization of the $ObsX_i$ idea discussed in §2.1 is to write dependencies of the form: *Observable*$(\psi(\bar{x}))$ $\{$*if* $\varphi(\bar{x})$ *then* $\sim cpd_1\}$, where w.l.o.g., φ and ψ 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". As noted in the introduction, this formulation leads to problems associated with *referential transparency* in first-order reasoning. In order to model the sensor accurately, we want to describe a belief state where *Vesper* = *Fiancee*(*Bond*), *Scanned*(*Vesper*) = *Scanned*(*Fiancee*(*Bond*)) = *true*, *Observable*(*MeasuredHeight*(*Vesper*)) = *true*, but *Observable*(*MeasuredHeight*(*Fiancee*(*Bond*))) = $false$. However, these statements will be inconsistent in any system based on first-order reasoning. This is because first-order logic enforces all terms to be referentially transparent, as a consequence of the application of the axiom of universal instantiation on the substitution axioms for equality. The axiom of universal instantiation states that if $\forall x\ \alpha(x)$ is true, then for any ground term t, $\alpha(t/x)$ (the version of $\alpha(x)$ with all free occurrences of x replaced by t) is also true. Different truth values for the observability statements above stem from the fact that the sensor knows who *Vesper* is, but not who *Fiancee*(*Bond*) is. This indicates that the knowledge of sensors and actuators needs to be taken into account while determining the possible effects of communicating with them.

To address this problem, we draw upon modal logic, where knowledge of the agent is taken into account (Levesque and Lakemeyer, 2000). The modal logic version of the axiom of universal instantiation states that if $\forall x\ \alpha(x)$ is true, then for any ground term t *whose value is known*, $\alpha(t/x)$ holds. In order to determine which terms are known, modal logic develops the notion of *rigid designators*: terms that have unique interpretations in all possible worlds according to the agent's knowledge. Our formulation ensures that only the modal logic version of the axiom of universal instantiation is used with statements involving observability and doability. In practice, we use a modal logic of observations, so that a term is considered to be known to a system iff its value is predefined or has been observed. Our framework to ensure that (a) observations are true statements and are thus suitable for conditioning, and (b) terms used in the interface with actuators have well-defined values.

3.2 FORMAL SPECIFICATIONS

We begin our formal solution with elementary descriptions of physical sensors and actuators that clarify the types of rigid designators that sensors (actuators) can provide (act upon). We assume that the vocabulary always includes, as rigid designators, constant symbols for elements of standard types such as natural numbers and strings; e.g., the constant symbol 1 represents the natural number 1 in every possible world. We also assume that standard mathematical and string operations have fixed interpretations and that terms that are applications of fixed functions on rigid designators are therefore also rigid designators (e.g., $1 + 2$). As explained in the following sections, observations may add to the set of rigid designators.

Any physical sensor can be described in terms of the types of symbols it can generate and the type signatures of the properties that it can report:

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 for the observation values that* S *produces and and* τ_S *denotes the type of new symbols that* S *may generate.*

E.g., an X-ray scanner can be specified as: $scanner = \langle \langle$*PersonRef, Length*$\rangle, \langle$*PersonRef*$\rangle \rangle$. Such a scanner can generate new symbols of type *PersonRef* and returns observation tuples of the type $\langle$*PersonRef*, *Length*$\rangle$; elements of type *Length* are real numbers with units (not terms such as *Height*(*Bond*)). Note that such a specification indicates that rigid designators of type $\bar{T}_S$ and τ_s are meaningful to the sensor, but arbitrary terms of the same types may not be.

A physical actuator can be specified similarly:

Definition 3. *An* actuator specification A *is a tuple of types*

$\bar{T}_A$ *denoting the types of its arguments.*

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

Let B be a set of beliefs, U be a set of actions, and O be a set of observations. A *strategy tree* $T_{B,U,O} = \langle \ell_U, \ell_O \rangle$ is defined by a mapping $\ell_U : B \rightarrow 2^U$, which defines the set of possible actions at each belief state, and a mapping $\ell_O : B, U \rightarrow 2^O$, which defines the set of possible observations when an action is applied on a belief state. Given a set of sensors $\mathcal{S}$ and actuators $\mathcal{A}$, an *actuator mapping* $\alpha_{\mathcal{A}} : U \rightarrow \mathcal{A}$ denotes the actuator responsible for executing each $u \in U$ and a *sensor mapping* $\sigma_{\mathcal{S}} : O \rightarrow \mathcal{S}$ denotes the sensor responsible for generating o. We drop the subscripts $\mathcal{S}$ and $\mathcal{A}$ when they are clear from context.

Definition 4. *A term is a* rigid designator *in a belief state b if it has a unique value in all possible worlds that have nonzero probability (in a discrete setting) or nonzero density (in a continuous setting) in b.*

Definition 5. *A strategy tree $T_{B,U,O} = \langle \ell_U, \ell_O \rangle$ is well-defined w.r.t. a set of sensor specifications $\mathcal{S}$, actuator specifications $\mathcal{A}$, a sensor mapping σ, and an actuator mapping α iff for every $b \in B$:*

- *Every $u \in \ell_U(b)$ uses as terms only the rigid designators of type $\bar{T}_A$ in b, where $A = \alpha(u)$.*
- *For every $u \in \ell_U(b)$, every $o \in \ell_O(b, u)$ is either the observation of symbol of type τ_S or uses as terms only the rigid designators of type $\bar{T}_S$ in b, where $S = \sigma(o)$.*

In other words, a strategy tree is well-defined iff the actions and observations that it specifies at each step are truly possible. A framework for solving OUPOMDPs therefore needs to ensure that the strategy trees it defines and uses in its solution algorithms are well-defined w.r.t. the given sensors and actuators. We now present our formulation of sensors and actuators for achieving this objective.

3.3 MODELING SENSORS

In this section we describe our formulation for a sensor $S = \langle \bar{T}_S, \tau_S \rangle$. Recall that τ_S is the set of symbols generated by S. By definition, such symbols evaluate to themselves (just like integers) and are therefore rigid designators. We specify 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 symbols of type τ_S includes an origin function τ_S_time which maps the symbol to the time when it was generated.

```
0  #Person ~ Poisson[10]; LocKnownDuration=4;
1  #PersonRef(Src = p, PersonRef_Time=t) {
2     if AtScanner(t)=p ~Bernoulli(0.5)
3     else = 0};
4  observableType(PersonRef);
5  observable(MeasuredHt(p_ref, t))=
6     (AtScanner(t) == Src(p_ref));
7  MeasuredHt(p_ref, t) ~ Normal[Ht(Src(p_ref)), 5];
8  Ht(prsn) ~ Normal[160, 30];
9
10 decision apply_TakePhoto(Orientation o, Distance d,
11     Timestep t);
12 PictureTaken(Person p, Timestep t){
13    if exists d, o RelativeDistance(TrueLoc(p, t)) == d &
14    RelativeOrientation(TrueLoc(p, t)) == o &
15    apply_takePhoto(o, d) then = true
16    else = False};
17
18 //Entrance model
19 AtScanner(t) ~ UniformChoice({Person prsn:
20   !Entered(prsn, t)});
21 Entered(prsn, t){
22   if t>0 & (AtScanner(t-1)=prsn) then = true
23   elseif t>0 then = Entered(prsn, t-1)
24   elseif t=0 then = false};
25 TrueLoc(Person p, t) ~ MovementModel(TrueLoc(p, t-1));
26 Location(p_ref, t) {
27   if PersonRef_Time(p_ref) < Horizon+1 &
28   & t<= PersonRef_Time(p_ref)+LocKnownDuration
29     then ~ TrueLoc(Src(p_ref, t))
30     else = null};
```

Figure 2: A DTBLOG model for the airport domain

- A dependency for *Observable*$(V_S(\bar{T}_S, \textit{Timestep}))$, indicating the factors influencing the probability that S generates an observation.

For ease in representation, we also allow syntax for capturing sensors that return values of functions declared in the model's vocabulary, rather than the default V_S predicate. Such observations also create rigid designators. 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*$_{scanner}(p, t)$). Lines 4-6 in Fig. 2 capture the DTBLOG specification for such a scanner. If the agent receives an observation of the form *MeasuredHt*$_{scanner}(\textit{pref}_{10,1}, 7) = 171cm$, then *MeasuredHt*$_{scanner}(\textit{pref}_{10,1}, 7)$ gets a unique interpretation, and thus becomes a rigid designator.

3.3.1 GENERATION OF OBSERVATIONS

Intuitively, if the environment is in state q at timestep t, and *Observable*$(\varphi(\bar{x}))$ (or *ObservableType*(τ_S)) is true in q, then the value of $\varphi(\bar{x})$ (or the symbols of type τ_S generated at a timestep t) must be obtained as evidence at timestep t. We operationalize this intuition through two types of observations in DTBLOG: observations of the symbols generated by sensors, and observations about properties of the observed symbols.

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. Suppose the model includes an observability declaration of

the form *ObservableType*(τ_S). For every possible world w at timestep t, this declaration results in the generation of a set evidence statement of the following form:

obs $\{\tau_S$ c:τ_S_time(c) == t$\}$ = $\{c_{t,1}, \ldots, c_{t,k(t)}\}$;

where $k(t) = \#\{\tau_S \quad x : \tau_S\_time(x) == t\}$ in w. Such a set evidence statement models the observation of a set of symbols generated by the sensor S. The semantics of BLOG ensure that $c_1, \ldots, c_{k(t)}$ are interpreted as distinct objects; e.g., the scanner declaration in Fig. 2 results in the generation of a set of *PersonRef* objects at each timestep. The number of such symbols is given by the CPD for the number statement for *PersonRef*. At timestep 10, this observation could be:

obs {PersonRef x: PersonRef_Time(x)==10} = {*pref*$_{10,1}$};

Relational observations For every true *Observable*($\varphi(\bar{x})$) atom in a state, the DTBLOG engine creates an observation statement of the form:

obs $\varphi(\bar{x})$ = v;

where all arguments and the value v are rigid designators. E.g., if in a possible world *Observable*(*MeasuredHt_scanner*(*pref*$_{10,1}$, 10))=*true* and *MeasuredHt_scanner*(*pref*$_{10,1}$, 10)=171*cm*, the generated relational observation would be:

obs MeasuredHt_scanner(*pref*$_{10,1}$, 10) = 171cm;

Each argument in such evidence statements has to be a rigid designator. Returning to the informal example described in the introduction, suppose Bond and Vesper were generated PersonRefs. The DTBLOG engine will not generate an observation of the form *MeasuredHeight*(*Fiancee*(*Bond*), t) = 150*cm* if the value of *Fiancee*(*Bond*) has not been observed (unless Fiancee is a fixed function, which would be quite unusual), even when *MeasuredHeight*() is observable for all PersonRefs. Thus, the agent being modeled will not "expect" an observation of the form "*MeasuredHeight*(*Fiancee*(*Bond*), t)=150*cm*". On the other hand, if the agent receives an observation of the form *Fiancee*(*Bond*)=*Vesper*, the term on the left becomes a rigid designator and future user-provided observations and system-generated decisions may use the term *Fiancee*(*Bond*).

We summarize our formulation of sensor models by noting that the set of all observations corresponding to a sensor S, $\sigma^{-1}(S)$, consists of the set evidence statements of type τ_S and value evidence statements for the predicate V_S (or its functional representation as noted above).

3.4 MODELING ACTUATORS

We represent an actuator $A = \bar{T}_A$ in a DTBLOG model as a decision function $apply\_A(\bar{T}_A, t)$. Decision functions are declared using the keyword *decision*; e.g., an actuated camera *TakePhoto* = $\langle$*Orientation*, *Length*$\rangle$ can be specified as:

decision apply_TakePhoto(Orientation, Distance, Timestep);

The space of all possible actions in a belief state could be defined using all possible substitutions of the arguments of decision variables with terms. However, without any further restrictions, this would lead to unintended situations where the user provides a decision of the form:

apply_TakePhoto(Orientation(Loc(Src(*pref*$_{10,1}$),t)),
DistanceTo(Loc(Src(*pref*$_{10,1}$),t)),t)=true;

even when *Loc*(*Src*(*pref*$_{10,1}$),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 objective was to determine the current location of the person who generated *pref*$_{10,1}$ and take their photo. We eliminate such situations by allowing only rigid designators as arguments of decision functions. Lines 10-16 in Fig. 2 capture the DTBLOG specification for a camera. The remainder of Fig. 2 completes the specification with dependencies for the persons' locations and their movement through the scanner. Since a *PersonRef* is generated when the someone is at the scanner, the location (of the person) corresponding to a *PersonRef* is known for a brief period.

We summarize our formulation of actuator models by noting that the set of all decisions corresponding to an actuator A, $\alpha^{-1}(A)$, consists of decision functions of the form $apply\_A(\bar{T}_A)$.

3.5 OUPOMDP DEFINED BY A DTBLOG MODEL

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 $\langle X, U, O, T, \Omega, R\rangle$ where the set of states X is the set of all timestep-indexed states corresponding to the possible worlds of vocabulary $\mathcal{V}_M$. U is the set of all instantiated decision functions corresponding to $\mathcal{A}$ that are allowed in some state $x \in X$ and O is the set of all instantiated functions corresponding to sensor specifications in $\mathcal{S}$ that are allowed in some state $x \in X$. The transition function T and observation function Ω are defined by the probabilistic dependency statements in M. This formulation does not place any requirements on distributions of the values of observations or the effects of actions. Every BLOG model must include dependencies for every declared function; the dependencies for the values of sensor-predicates and fluents affected by decisions can be defined to capture arbitrary physical processes.

Strategy tree generated by a DTBLOG model DTBLOG represents belief states using collections of sampled, possible states. BLOG's existing sampling subroutines are used to sample possible worlds corresponding to the initial state specified in the BLOG model. The semantics developed above implicitly define the strategy tree for a DTBLOG model: the application of a decision on a belief updates each possible world in the belief to the next timestep using the stated dependencies. When the belief

state is updated w.r.t. to a decision, the DTBLOG engine generates the set of observations corresponding to each possible updated state.

The following results follow from the semantics above.

Lemma 1. *The semantics of DTBLOG ensure that in a belief b, (a) every generated observation statement o is either a set evidence statement for symbols of type $\tau_{\sigma(o)}$ or an observation statement utilizing only rigid designators of type specified by $\bar{T}_{\sigma(o)}$, and (b) every possible decision u uses only arguments that are rigid designator tuples of the type $\bar{T}_{\alpha(u)}$.*

Theorem 1. *Let $M(\mathcal{S}, \mathcal{A})$ be a DTBLOG model defined using sensor and actuator specifications $\mathcal{S}$ and $\mathcal{A}$ respectively, such that $\sigma_{\mathcal{S}}$ and $\alpha_{\mathcal{A}}$ are its sensor and actuator mappings. If M satisfies BLOG's requirements for well-defined probabilistic models then it generates a well-defined strategy tree w.r.t. $\mathcal{S}, \mathcal{A}, \sigma_{\mathcal{S}}$, and $\alpha_{\mathcal{A}}$.*

3.6 MODELING REAL-WORLD SITUATIONS

We now show that various non-trivial modes of sensing and acting can be expressed easily in the presented framework.

Observations of composed functions Consider a human operator who reads the passport of a passenger immediately after she exits the height scanner, and reports the date of birth along with the person reference generated by the height scanner. Intuitively, the human operator provides observations of a function composition: *DOB*(*PassportID*(*Src*(*PersonRef*))), where *DOB* maps a passport number to the date of birth mentioned within. This appears to be a problem since the use of function application terms is specifically disallowed in our framework unless each such term has been observed. However, the situation can be modeled by defining the human operator as a sensor $\langle\langle$*PersonRef,Date*$\rangle, \langle\rangle\rangle$, that reports observations about the derived function *DOBonPPofSrc*(x)=*DOB*(*PassportID*(*Src*(x))). In fact, our formulation correctly ensures that the agent will not expect an observation of *DOB*(*PassportID*(*Src*(x))) from a sensor that reports the *DOB*(y) function (perhaps a swipe-through scanner).

Actions on sensor-generated symbols In the real world, agents need to act upon objects that are detected through their sensors. Clearly, actuators can be specified to directly take inputs of the type generated by a sensor. Most commonly however, physical actuators such as a robot's gripper cannot act upon symbols. In such situations, sensor-generated symbols can be used 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 in addition to their measured heights. Let the functions *RelativeOrientation* and *CamDistance* map positions to the orientation and distance relative to the camera, respectively. We can define a camera action that takes a snapshot given a *PersonRef* as follows:

```
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*().

Actuators that gain knowledge Although this framework cannot capture completely the beliefs of actuators or sensors with their own reasoning abilities, it can model actuators and sensors whose knowledge evolves. Consider an actuator that can dial phone numbers. Suppose this actuator models a human operator, who can read the phone book to obtain the phone number for any name. Our system correctly allows decisions such as *apply_Dial*(*phoneNumber*(*"John"*), t) only if an observation of *PhoneNumber*(*"John"*) is received. However, they should also be allowed if the operator has been able to look up John's phone number in the phone book. The non-trivial part in designing such a system is to model the growth in knowledge of the operator.

Such a phone operator can be specified as $\langle$*String*, *Time*$\rangle$. The corresponding decision variable takes the form *apply_Dial*(s, t). The restrictions developed above, on possibly substitutions for s, apply here. The effects of this action are determined by a predicate:

```
dialNumber(n, t) {
    if exists String s apply_Dial(s, t)
        and n == lookedUpNumber(s, t) then = true
    else = false};
```

where *lookedUpNumber*(s, t) is either the number represented by s if s is the string representation of a number, or, if s is a name, then it is the looked up number for that name according to a certain distribution:

```
lookedUpNumber(String s, Time t) {
    if strToNumber(s) != null then = strToNumber(s)
    else if exists t'<t operatorLookedUp(s, t')
        then ~ phoneBookCPD(s)
    else = null };
```

With this formulation, the application of decisions like *apply_dialNumber*(*"John"*) will result in a correctly updated belief state, contingent upon whether or not the operator performed a lookup.

4 SOLVING OUPOMDPS: OU-PBVI

In order to illustrate the efficacy of our framework, we developed and implemented an open-universe version of point-based value iteration (PBVI) (Pineau et al., 2003). OU-PBVI is designed to handle the main aspects of OUPOMDPs that prevent a direct application of POMDP algorithms: (a) belief states are infinite dimensional objects, and (b) the set of possible observations is unbounded. Thus, standard POMDP algorithms, which rely upon carrying out analytical backup operations using the domain's

transition function cannot be applied in general (although closed form backups for special cases may be possible). The following description focuses upon the key enhancements made to PBVI; a description of PBVI itself can be found in the cited reference.

PBVI restricts the Bellman backup of the value function for POMDPs to a finite set of sampled belief states, B. The $\|X\|$-dimensional $\boldsymbol{\alpha}$ vector for a policy represents its value function in terms of the expected value of following the policy for each state $x \in X$. PBVI approaches construct $t+1$ step policies by computing the $t+1$ step value function for each $b \in B$:

$$V^{t+1}(b) = max_u\{E_b[r(s,u)] + \gamma E_{\Omega(o|b,u)}[V^t_{max}(b^u_o)]\},$$

where $V^t_{max}(b^u_o) = max_{\boldsymbol{\alpha}\in V_t}\boldsymbol{\alpha}\cdot b^u_o$, and b^u_o is the belief state after action u is applied in b and observation o is obtained. $V^{t+1}(b)$ represents a 1-step backup of V for b.

In order to apply these ideas to OUPOMDPs, OU-PBVI represents belief states as sampled sets of possible worlds. Action application and the generation of observations proceeds as discussed in detail in the preceding sections. Thus, for each sampled state in a belief state b, action application results in a set of observations corresponding to the sampled states in b. We use a particle filter with N_p particles for belief propagation, and replace expectations by sample averages (Srivastava et al., 2012). In addition, we evaluate $\boldsymbol{\alpha}$ vectors lazily. In our implementation, the number of keys (possible worlds) in $\boldsymbol{\alpha}$ may reach a bound of $N_p \cdot \|B\|$. We also store the policy corresponding to each $\boldsymbol{\alpha}$ vector. This allows us to dynamically compute missing components of the vector, as needed. More precisely, during backup, if the value of a policy π is required for a state that is not in the current set of states captured by $\boldsymbol{\alpha}_\pi$, that state is added to $\boldsymbol{\alpha}_\pi$ and its value is computed by carrying out N_E simulations of the execution of that policy. During each simulation, after each step of policy execution and observation generation, if the resulting sampled state x is present in $\boldsymbol{\alpha}_{\pi'}$ where π' is the subpolicy of π that remains to be executed, $\boldsymbol{\alpha}_{\pi'}(x)$ is used and the simulation terminates. The estimates of policy value functions converge to the true value functions as the $N_p, N_E \to \infty$.

5 TEST PROBLEM AND RESULTS

We conducted experiments to investigate whether point-based approaches could be used for solving OUPOMDPs. As a test problem we developed an open-universe version of the Tiger problem. The agent is surrounded by 4 zones with an unknown, unbounded number of tigers who may move among zones at each timestep. Multiple tigers may be in a zone and the objective is to enter a zone without a tiger. The agent has two actions, a *listen*(*Timestep* t) action that allows it to make inaccurate observations about the growls made by tigers at timestep t, and an *enter*(*Zone* z, *Timestep* t) action which it can use to enter a zone. When *listen* is applied, the agent obtains a growl from each tiger with probability 0.5.

```
observableType(Growl);
#Growl(Source = m, Time_Growl = t) {
    if apply_listen(t-1) then ~ Bernoulli(0.5)
    else = 0};
```

The listen action also gives a noisy estimate of the zones from which growls came. If a growl is made by a tiger in z, the probability of observing that a growl was made at z is 0.75, and that of observing that a growl was made at each of the zones $(z+1 \bmod 4)$ and $(z-1 \bmod 4)$ is 0.25. At each timestep, each tiger independently either stays in its zone with probability 0.4 or moves to each of the neighboring zones with probability 0.3. The agent receives a reward 10 for entering a door without tiger, -1 for listening, and -100 for entering a zone with a tiger. The number of tigers follows a Poisson(1) distribution. The objective is to find a policy for the belief state specified in the DTBLOG model. The unknown, *unbounded* numbers of tigers and *relevant* observations, as well as the independent movements of tigers make this a difficult OUPOMDP that cannot be expressed using existing approaches.

For our evaluation, we fixed $N_E = 100, \gamma = 0.9$ and the time horizon at 5. Fig. 3 shows the values of the obtained policies for varying values of $\|B\|$ and N_p, averaged over 10 different runs of the algorithm. To compute estimates of policy values, we carried out N_E simulations of the computed policy on each of 5000 samples for the initial state. The results show that the variance in policy value-estimates decrease as N_p and $\|B\|$ increase.

Since no other existing approach addresses OUPOMDPs, we used a *belief-state query* (BSQ) policy (Srivastava et al., 2012) as a baseline. BSQ policies map first-order probabilistic queries to belief states. Although such policies are more succinct than observation-history policies, evaluating them is non-trivial because the action to be applied at each step depends on the posterior probability of a first-order query rather than just the observation history. Algorithms for policy evaluation are discussed in detail in the cited reference. For our experiments we used the following, intuitively simple BSQ policy:

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 found that $\theta = 0.9$ resulted in the highest expected value for this BSQ policy, shown using the dashed line in Fig. 3. Our experiments show that OU-PBVI's computed policies approach the best value of our hand-written, parameterized BSQ policy as N_p and $\|B\|$ increase.

The average runtimes for OU-PBVI ranged from 420s for 8 beliefs and 200 particles to 106, 800s for 1000 particles and 128 beliefs. At least 80% of the time was spent in proba-

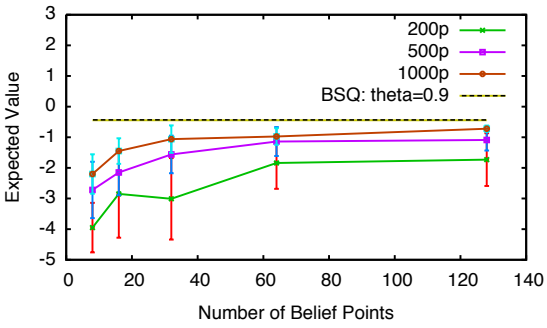

Figure 3: Expected values for OU-PBVI. Solid lines represent different numbers of particles used for the belief state representation (see text for details).

bilistic inference for carrying out belief propagation. Since we utilize PPLs to express OUPOMDPs, our approach will automatically scale with improvements in their inference engines. Indeed, using compiler techniques for improving inference in PPLs is an active area of research, and has shown speed-ups of up to 200x in preliminary experiments. We expect the runtimes of our algorithm to reduce to a fraction of the current estimates as a result of such advances.

These experiments demonstrate that our framework can be used to effectively express OUPOMDPs and solve them.

6 DISCUSSION

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, they cannot be used in open-universe probabilistic languages due to the requirement of reifying possible worlds and terms as objects in a universe. Further, they do not address the problems 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 (SSAs) (Reiter, 2001) with a significant enhancement: they allow compact expression of the so-called factored representations that are difficult to express using SSAs (Sanner and Kersting, 2010). Moreover, usually employed assumptions like having a "closed initial database" in that line of work preclude the possibility of expressing identity uncertainty: distinct terms like *Vesper* and *Fiancee*(*Bond*) can never represent the same object. Sanner and Kersting (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. The RDDL language (Sanner, 2010) used in recent probabilistic planning competitions can also express closed-universe POMDPs as relational extensions of DBNs, under the assumptions that a fixed set of predicates will be observed at every time step and that all ground terms are unique and known. Wang and Khardon (2010) present a relational representation for closed-universe POMDPs where action arguments have to be in a known 1-1 mapping with actual objects in the universe. Kaelbling and Lozano-Pérez (2013) present an approach where action specifications include preconditions in the form of belief-state fluents. However, the solution approach requires action-specific regression functions over the probabilities of such queries. The approaches discussed so far assume that a POMDP definition is available. Recent work by Doshi-Velez (2010) addresses the problem of learning the transition and observation distributions for an *unfactored* POMDP with a potentially unbounded number of states.

In contrast to these approaches, our formulation allows an agent to plan and act upon objects *discovered* through its sensors. We presented the first framework for accurately expressing OUPOMDPs and solving them. The central idea of our solution is that representing observations and decisions using terms with unique meanings clarifies communication without being restrictive. We utilized this idea to construct strategy trees reflecting the true capabilities of an agent—something that could not be achieved by extending the existing formalisms. We also showed that this framework facilitates general algorithms for solving a large class of decision problems that capture real-world situations and could not previously be expressed or solved. A number of directions exist for future work. The notion of high-level actions can be developed further. For instance, one could define an action that determines the maximum likelihood estimate for the position of the person who generated a reference, and takes a picture of that location. Such actions have to be specified outside the DTBLOG model since they need to execute queries on the model to construct their arguments. However, probabilistic effects of such actions have to be defined in the model in a manner consistent with their external definitions.

Acknowledgments

This research was supported in part by DARPA contracts W31P4Q-11-C-0083 and FA8750-14-C-0011, ONR grant N00014-12-1-0609 and ONR MURI award ONR-N00014-13-1-0341.

References

Astrom, K. J. 1965. Optimal control of Markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications* 10:174–205.

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

Davis, E. 2005. Knowledge and communication: A first-order theory. *AIJ* 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.

Doshi-Velez, F. 2010. The infinite partially observable markov decision process. In *Neural Information Processing Systems (NIPS)*, volume 22.

Goodman, N. D.; Mansinghka, V. K.; Roy, D. M.; Bonawitz, K.; and Tenenbaum, J. B. 2008. Church: A language for generative models. In *UAI-08*.

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. *AIJ* 101(1-2):99–134.

Koller, D.; McAllester, D.; and Pfeffer, A. 1997. Effective Bayesian inference for stochastic programs. In *AAAI-97*.

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. IJCAI-05*, 1352–1359.

Milch, B. C. 2006. *Probabilistic models with unknown objects*. Ph.D. Dissertation, UC Berkeley.

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. IJCAI-87*, 867–874.

Pineau, J.; Gordon, G.; Thrun, S.; et al. 2003. Point-based value iteration: An anytime algorithm for POMDPs. In *Proc. IJCAI-03*.

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

Russell, S. J., and Norvig, P. 1995. *Artificial Intelligence - A Modern Approach*. Prentice Hall.

Sanner, S., and Kersting, K. 2010. Symbolic dynamic programming for first-order POMDPs. In *Proc. AAAI-10*.

Sanner, S. 2010. Relational dynamic influence diagram language (RDDL): Language description. `http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/RDDL.pdf`.

Srivastava, S.; Cheng, X.; Russell, S.; and Pfeffer, A. 2012. First-order open-universe POMDPs: Formulation and algorithms. Technical report, EECS-2013-243, EECS Department, UC Berkeley.

Wang, C., and Khardon, R. 2010. Relational partially observable MDPs. In *Proc. AAAI-10*.