BLOG: Probabilistic Models with Unknown Objects*

Brian Milch, Bhaskara Marthi, Stuart Russell, David Sontag, Daniel L. Ong and **Andrey Kolobov**
Computer Science Division
University of California
Berkeley, CA 94720-1776
{milch, bhaskara, russell, dsontag, dlong, karaya1}@cs.berkeley.edu

Abstract

This paper introduces and illustrates BLOG, a formal language for defining probability models over worlds with unknown objects and identity uncertainty. BLOG unifies and extends several existing approaches. Subject to certain acyclicity constraints, every BLOG model specifies a unique probability distribution over first-order model structures that can contain varying and unbounded numbers of objects. Furthermore, complete inference algorithms exist for a large fragment of the language. We also introduce a probabilistic form of Skolemization for handling evidence.

1 Introduction

Human beings and AI systems must convert sensory input into some understanding of what's out there and what's going on in the world. That is, they must make inferences about the objects and events that underlie their observations. No pre-specified list of objects is given; the agent must infer the existence of objects that were not known initially to exist.

In many AI systems, this problem of unknown objects is engineered away or resolved in a preprocessing step. However, there are important applications where the problem is unavoidable. *Population estimation*, for example, involves counting a population by sampling from it randomly and measuring how often the same object is resampled; this would be pointless if the set of objects were known in advance. *Record linkage*, a task undertaken by an industry of more than 300 companies, involves matching entries across multiple databases. These companies exist because of uncertainty about the mapping from observations to underlying objects. Finally, *multi-target tracking* systems perform *data association*, connecting, say, radar blips to hypothesized aircraft.

Probability models for such tasks are not new: Bayesian models for data association have been used since the 1960s [Sittler, 1964]. The models are written in English and mathematical notation and converted by hand into special-purpose code. In recent years, *formal representation languages* such as graphical models [Pearl, 1988] have led to general inference algorithms, more sophisticated models, and automated model selection (structure learning). In Sec. 7, we review several *first-order probabilistic languages* (FOPLs) that explicitly represent objects and the relations between them. However, most FOPLs only deal with fixed sets of objects, or deal with unknown objects in limited and *ad hoc* ways. This paper introduces BLOG (Bayesian LOGic), a compact and intuitive language for defining probability distributions over outcomes with varying sets of objects.

We begin in Sec. 2 with three example problems, each of which involves possible worlds with varying object sets and identity uncertainty. We describe generative processes that produce such worlds, and give the corresponding BLOG models. Sec. 3 observes that these possible worlds are naturally viewed as model structures of *first-order logic*. It then defines precisely the set of possible worlds corresponding to a BLOG model. The key idea is a generative process that constructs a world by adding objects whose existence and properties depend on those of objects already created. In such a process, the existence of objects may be governed by many random variables, not just a single population size variable. Sec. 4 discusses how a BLOG model specifies a probability distribution over possible worlds.

Sec. 5 solves a previously unnoticed "probabilistic Skolemization" problem: how to specify evidence about objects—such as radar blips—that one didn't know existed. Finally, Sec. 6 briefly discusses inference in unbounded outcome spaces, stating a sampling algorithm and a completeness theorem for a large class of BLOG models, and giving experimental results on one particular model.

2 Examples

In this section we examine three typical scenarios with unknown objects—simplified versions of the population estimation, record linkage, and multitarget tracking problems mentioned above. In each case, we provide a short BLOG model that, when combined with a suitable inference engine, constitutes a working solution for the problem in question.

Example 1. *An urn contains an unknown number of balls—say, a number chosen from a Poisson distribution. Balls are equally likely to be blue or green. We draw some balls from the urn, observing the color of each and replacing it. We cannot tell two identically colored balls apart; furthermore, observed colors are wrong with probability 0.2. How many balls are in the urn? Was the same ball drawn twice?*

The BLOG model for this problem, shown in Fig. 1, describes a stochastic process for generating worlds. The first 4

*This work was supported by DARPA under award 03-000219, and by an NSF Graduate Research Fellowship to B. Milch.

```
1 type Color;  type Ball;  type Draw;

2 random Color TrueColor(Ball);
3 random Ball BallDrawn(Draw);
4 random Color ObsColor(Draw);

5 guaranteed Color Blue, Green;
6 guaranteed Draw Draw1, Draw2, Draw3, Draw4;

7 #Ball ~ Poisson[6]();

8 TrueColor(b) ~ TabularCPD[[0.5, 0.5]]();

9 BallDrawn(d) ~ Uniform({Ball b});

10 ObsColor(d)
11     if (BallDrawn(d) != null) then
12         ~ TabularCPD[[0.8, 0.2], [0.2, 0.8]]
13             (TrueColor(BallDrawn(d)));
```

Figure 1: BLOG model for the urn-and-balls scenario of Ex. 1 with four draws.

lines introduce the types of objects in these worlds—colors, balls, and draws—and the functions that can be applied to these objects. Lines 5–7 specify what objects may exist in each world. In every world, the colors are blue and green and there are four draws; these are the *guaranteed* objects. On the other hand, different worlds have different numbers of balls, so the number of balls that exist is chosen from a prior—a Poisson with mean 6. Each ball is then given a color, as specified on line 8. Properties of the four draws are filled in by choosing a ball (line 9) and an observed color for that ball (lines 10–13). The probability of the generated world is the product of the probabilities of all the choices made.

Example 2. *We have a collection of citations that refer to publications in a certain field. What publications and researchers exist, with what titles and names? Who wrote which publication, and which publication does each citation refer to? For simplicity, we just consider the title and author-name strings in these citations, which are subject to errors of various kinds, and we assume only single-author publications.*

Fig. 2 shows a BLOG model for this example, based on the model in [Pasula *et al.*, 2003]. The BLOG model defines the following generative process. First, sample the total number of researchers from some distribution; then, for each researcher r, sample the number of publications by that researcher. Sample the publications' titles and researchers' names from appropriate prior distributions. Then, for each citation, sample the publication cited by choosing uniformly at random from the set of publications. Sample the title and author strings used in each citation from string corruption models conditioned on the true attributes of the cited publication; finally, generate the full citation string according to a formatting model.

Example 3. *An unknown number of aircraft exist in some volume of airspace. An aircraft's state (position and velocity) at each time step depends on its state at the previous time step. We observe the area with radar: aircraft may appear as identical blips on a radar screen. Each blip gives the approximate position of the aircraft that generated it. However, some blips may be false detections, and some aircraft may not be detected. What aircraft exist, and what are their trajectories? Are there any aircraft that are not observed?*

The BLOG model for this scenario (Fig. 3) describes the following process: first, sample the number of aircraft in the area. Then, for each time step t (starting at $t = 0$), choose the state (position and velocity) of each aircraft given its state at time $t - 1$. Also, for each aircraft a and time step t, possibly generate a radar blip r with $\mathsf{Source}(r) = a$ and $\mathsf{Time}(r) = t$. Whether a blip is generated or not depends on the state of the aircraft—thus the number of objects in the world depends on certain objects' attributes. Also, at each step, generate some false alarm blips r' with $\mathsf{Source}(r') = \mathsf{null}$. Finally, sample the position for each blip given the true state of its source aircraft (or using a default distribution for a false-alarm blip).

```
guaranteed Citation Cite1, Cite2, Cite3, Cite4;

#Researcher ~ NumResearchersDistrib();
#Publication: (Author) -> (r) ~ NumPubsDistrib();

Title(p) ~ TitlePrior();
Name(r) ~ NamePrior();

PubCited(c) ~ Uniform({Publication p});

TitleString(c) ~ TitleObs(Title(PubCited(c)));
AuthorString(c) ~ AuthorObs(Name(Author(PubCited(c))));

CitString(c) ~ CitDistrib(TitleString(c),AuthorString(c));
```

Figure 2: BLOG model for Ex. 2 with four observed citations (type and function declarations are omitted).

3 Syntax and Semantics: Possible Worlds

3.1 Outcomes as first-order model structures

The possible outcomes for Ex. 1 through Ex. 3 are structures containing many related objects, which we will treat formally as *model structures* of *first-order logic*. A model structure provides interpretations for the symbols of a first-order language. (Usually, first-order languages are described as having predicate, function, and constant symbols. For conciseness, we view all symbols as function symbols; predicates are just functions that return a Boolean value, and constants are just zero-ary functions.)

For Ex. 1, the language has function symbols such as $\mathsf{TrueColor}(b)$ for the true color of ball b; $\mathsf{BallDrawn}(d)$ for the ball drawn on draw d; and $\mathsf{Draw1}$ for the first draw. (Statisticians might use indexed families of random variables such as $\{\mathsf{TrueColor}_b\}$, but this is mainly a matter of taste.)

```
type Aircraft;  type Blip;

random R6Vector State(Aircraft, NaturalNum);
random R3Vector ApparentPos(Blip);

nonrandom NaturalNum Pred(NaturalNum) = Predecessor;

generating Aircraft Source(Blip);
generating NaturalNum Time(Blip);

#Aircraft ~ NumAircraftDistrib();

State(a, t)
  if t = 0 then ~ InitState()
  else ~ StateTransition(State(a, Pred(t)));

#Blip: (Source, Time) -> (a, t)
  ~ DetectionDistrib(State(a, t));

#Blip: (Time) -> (t)
  ~ NumFalseAlarmsDistrib();

ApparentPos(r)
  if (Source(r) = null) then ~ FalseAlarmDistrib()
  else ~ ObsDistrib(State(Source(r), Time(r)));
```

Figure 3: BLOG model for Ex. 3.

To eliminate meaningless random variables, we use a *typed*, *free* logical language, in which every object has a type and in which functions may be partial. Each function symbol f has a *type signature* $(\tau_0, \ldots, \tau_k)$, where τ_0 is the return type of f and $\tau_1, \ldots, \tau_k$ are the argument types. A partial function applied to arguments outside its domain returns the special value null, which is not of any type.

The truth of any first-order sentence is determined by a *model structure* for the corresponding language. A model structure specifies the *extension* of each type and the *interpretation* for each function symbol:

Definition 1. *A model structure ω of a typed, free, first-order language consists of an extension $[\tau]^\omega$ for each type τ, which may be an arbitrary set, and an interpretation $[f]^\omega$ for each function symbol f. If f has return type τ_0 and argument types $\tau_1, \ldots, \tau_k$, then $[f]^\omega$ is a function from $[\tau_1]^\omega \times \cdots \times [\tau_k]^\omega$ to $[\tau_0]^\omega \cup \{\mathsf{null}\}$.*

A BLOG model defines a probability distribution over a particular set of model structures. In Ex. 1, identity uncertainty arises because $[\mathsf{BallDrawn}]^\omega (\mathsf{Draw1})$ might be equal to $[\mathsf{BallDrawn}]^\omega (\mathsf{Draw2})$ in one structure but not another. The set of balls, $[\mathsf{Ball}]^\omega$, can also vary between structures.

3.2 Outcomes with fixed object sets

BLOG models for fixed object sets have five kinds of statements. A *type declaration*, such as the two statements on line 1 of Fig. 3, introduces a type. Certain types, namely $\mathsf{Boolean}$, $\mathsf{NaturalNum}$, $\mathsf{Integer}$, String, Real, and $\mathsf{R}k\mathsf{Vector}$ (for each $k \geq 2$) are already provided. A *random function declaration*, such as line 2 of Fig. 3, specifies the type signature for a function symbol whose values will be chosen randomly in the generative process. A *nonrandom function definition*, such as the one on line 4 of Fig. 3, introduces a function whose interpretation is fixed in all possible worlds. In our implementation, the interpretation is given by a Java class (`Predecessor` in this example). A *guaranteed object statement*, such as line 5 in Fig. 1, introduces and names some distinct objects that exist in all possible worlds. For the built-in types, the obvious sets of guaranteed objects and constant symbols are predefined. The set of guaranteed objects of type τ in BLOG model M is denoted $G_M(\tau)$. Finally, for each random function symbol, a BLOG model includes a *dependency statement* specifying how values are chosen for that function. We postpone further discussion of dependency statements to Sec. 4.

The first four kinds of statements listed above define a particular typed first-order language $\mathcal{L}_M$ for a model M. The set of *possible worlds* of M, denoted Ω_M, consists of those model structures of $\mathcal{L}_M$ where the extension of each type τ is $G_M(\tau)$, and all nonrandom function symbols (including guaranteed constants) have their given interpretations.

For each random function f and tuple of appropriately typed guaranteed objects $o_1, \ldots, o_k$, we can define a random variable (RV) $V_f[o_1, \ldots, o_k](\omega) \triangleq [f]^\omega (o_1, \ldots, o_k)$. The possible worlds are in one-to-one correspondence with full instantiations of these basic RVs. Thus, a joint distribution for the basic RVs defines a distribution over possible worlds.

3.3 Unknown objects

In general, a BLOG model defines a generative process in which objects are added iteratively to a world. To describe such processes, we first introduce *generating function declarations*, such as lines 5–6 of Fig. 3. Unlike other functions, generating functions such as Source or Time have their values set when an object is added. A generating function must take a single argument of some type τ (namely Blip in the example); it is then called a τ-generating function.

Generative steps that add objects to the world are described by *number statements*. For instance, line 11 of Fig. 3 says that for each aircraft a and time step t, the process adds some number of blips r such that $\mathsf{Source}(r) = a$ and $\mathsf{Time}(r) = t$. In general, the beginning of a number statement has the form: $\#\tau$: $(g_1, \ldots, g_k)$ -> $(x_1, \ldots, x_k)$ where τ is a type, $g_1, \ldots, g_k$ are τ-generating functions, and $x_1, \ldots, x_k$ are logical variables. (For types that are generated *ab initio* with no generating functions, the `() -> ()` is omitted, as in Fig. 1.) The inclusion of a number statement means that for each appropriately typed tuple of objects $o_1, \ldots, o_k$ (the appropriate types are the return types of $g_1, \ldots, g_k$), the generative process adds some number (possibly zero) of objects q of type τ such that $[g_i]^\omega (q) = o_i$ for $i = 1, \ldots, k$.

Object generation can even be recursive: for instance, in a model for phylogenetic trees, we could have a generating function Parent that takes a species and returns a species; then we could model speciation events with a number statement that begins: `#Species: (Parent) -> (p)`.

We can also view number statements more declaratively:

Definition 2. *Let ω be a model structure of $\mathcal{L}_M$, and consider a number statement for type τ with generating functions $g_1, \ldots, g_k$. An object $q \in [\tau]^\omega$* satisfies *this number statement applied to $o_1, \ldots, o_k$ in ω if $[g_i]^\omega (q) = o_i$ for $i = 1, \ldots, k$, and $[g]^\omega (q) = \mathsf{null}$ for all other τ-generating functions g.*

Note that if a number statement for type τ omits one of the τ-generating functions, then this function takes on the value null for all objects satisfying that number statement. For instance, Source is null for objects satisfying the false-detection number statement on line 13 of Fig. 3. Also, a BLOG model cannot contain two number statements with the same set of generating functions. This ensures that each object o has exactly one generation history, which can be found by tracing back the generating functions on o.

The set of possible worlds Ω_M is the set of model structures that can be constructed by M's generative process. To complete the picture, we must explain not only *how many* objects are added on each step, but also *what* these objects are. It turns out to be convenient to define the generated objects as follows: when a number statement with type τ and generating functions $g_1, \ldots, g_k$ is applied to generating objects $o_1, \ldots, o_k$, the generated objects are tuples $\{(\tau, (g_1, o_1), \ldots, (g_k, o_k), n) \ : \ n = 1, \ldots, N\}$, where N is the number of objects generated. Thus in Ex. 3, the aircraft are pairs $(\mathsf{Aircraft}, 1)$, $(\mathsf{Aircraft}, 2)$, etc., and the blips generated by aircraft are nested tuples such as $(\mathsf{Blip}, (\mathsf{Source}, (\mathsf{Aircraft}, 2)), (\mathsf{Time}, 8), 1)$. The tuple encodes the object's generation history; of course, it is purely internal to the semantics and remains invisible to the user.

The set of all objects (nested tuples) of type τ that can be generated from the guaranteed objects via finitely many recursive applications of number statements is called the *universe* of τ, and denoted $U_M(\tau)$. In each possible world, the extension of τ is some subset of $U_M(\tau)$.

Definition 3. *For a* BLOG *model M, the set of possible worlds Ω_M is the set of model structures ω of $\mathcal{L}_M$ such that:*

1. *for each type τ, $G_M(\tau) \subseteq [\tau]^\omega \subseteq U_M(\tau)$;*
2. *nonrandom functions have the specified interpretations;*
3. *for each number statement in M with type τ and generating functions $g_1, \ldots, g_k$, and each appropriately typed tuple of generating objects $(o_1, \ldots, o_k)$ in ω, the set of objects in $[\tau]^\omega$ that satisfy this number statement applied to these generating objects is $\{(\tau, (g_1, o_1), \ldots, (g_k, o_k), n) : n = 1, \ldots, N\}$ for some natural number N;*
4. *for every type τ, each element of $[\tau]^\omega$ satisfies some number statement applied to some objects in ω.*

Note that by part 3 of this definition, the number of objects generated by any given application of a number statement in world ω is a finite number N. However, a world can still contain infinitely many non-guaranteed objects if some number statements are applied recursively, or are triggered for every natural number (like the ones generating radar blips in Ex. 3).

With a fixed set of objects, it was easy to define a set of basic RVs such that a full instantiation of the basic RVs uniquely identified a possible world. To achieve the same effect with unknown objects, we need two kinds of basic RVs:

Definition 4. *For a* BLOG *model M, the set $\mathbf{V}_M$ of* basic random variables *consists of:*

- *for each random function f with type signature $(\tau_0, \ldots, \tau_k)$ and each tuple of objects $(o_1, \ldots, o_k) \in U_M(\tau_1) \times \cdots \times U_M(\tau_k)$, a* function application RV *$V_f[o_1, \ldots, o_k](\omega)$ that is equal to $[f]^\omega(o_1, \ldots, o_k)$ if $o_1, \ldots, o_k$ all exist in ω, and* null *(or* false *for Boolean RVs) otherwise;*
- *for each number statement with type τ and generating functions $g_1, \ldots, g_k$ that have return types $\tau_1, \ldots, \tau_k$, and each tuple of objects $(o_1, \ldots, o_k) \in U_M(\tau_1) \times \cdots \times U_M(\tau_k)$, a* number RV *$N_{(\tau, g_1, \ldots, g_k)}[o_1, \ldots, o_k](\omega)$ equal to the number of objects that satisfy this number statement applied to $o_1, \ldots, o_k$ in ω.*

Intuitively, each step in the generative world-construction process determines the value of a basic variable. The crucial result about basic RVs is the following:

Proposition 1. *For any* BLOG *model M and any complete instantiation of $\mathbf{V}_M$, there is at most one model structure in Ω_M consistent with this instantiation.*

Equating objects with tuples might seem unnecessarily complicated, but it becomes very helpful when we define a Bayes net over the basic RVs (which we do in Sec. 4.2). For instance, the sole parent of $V_{\mathsf{ApparentPos}}[(\mathsf{Blip}, (\mathsf{Source}, (\mathsf{Aircraft}, 2)), (\mathsf{Time}, 8), 1)]$ is $V_{\mathsf{State}}[(\mathsf{Aircraft}, 2), 8]$. It might seem more elegant to assign numbers to objects as they are generated, so that the extension of each type in each possible world would be simply a prefix of the natural numbers. Specifically, we could number the aircraft arbitrarily, and then number the radar blips lexicographically by aircraft and time step. Then we would have basic RVs such as $V_{\mathsf{ApparentPos}}[23]$, representing the apparent aircraft position for blip 23. But blip 23 could be generated by any aircraft at any time step. In fact, the parents of $V_{\mathsf{ApparentPos}}[23]$ would have to include all the $\#\mathsf{Blip}$ and State variables in the model. So, defining objects as tuples yields a much simpler Bayes net.

4 Syntax and Semantics: Probabilities

4.1 Dependency statements

Dependency and number statements specify exactly how the steps are carried out in our generative process. Consider the dependency statement for $\mathsf{State}(a, t)$ beginning on line 8 of Fig. 3. This statement is applied for every basic RV of the form $V_{\mathsf{State}}[a, t]$ where $a \in U_M(\mathsf{Aircraft})$ and $t \in \mathbb{N}$. If $t = 0$, the conditional distribution for $\mathsf{State}(a, t)$ is given by the *elementary CPD* `InitState`; otherwise it is given by the elementary CPD `StateTransition`, which takes State(a, Pred(t)) as an argument. These elementary CPDs define distributions over objects of type R6Vector (the return type of State). In our implementation, elementary CPDs are Java classes with a method `getProb` that returns the probability of a particular value given a list of CPD arguments.

A dependency statement begins with a function symbol f and a tuple of logical variables $x_1, \ldots, x_k$ representing the arguments to this function. In a number statement, the variables $x_1, \ldots, x_k$ represent the generating objects. In either case, the rest of the statement consists of a sequence of *clauses*. When the statement is not abbreviated, the syntax for the first clause is `if` *`cond`* `then` $\sim$ *`elem-cpd`*`(`*`arg1`*`, ...,` *`argN`*`)`. The *cond* portion is a formula of $\mathcal{L}_M$ (in which only $x_1, \ldots, x_k$ can occur as free variables) specifying the condition under which this clause should be used to sample a value for a basic RV. More precisely, if the possible world constructed so far is ω, then the applicable clause is the *first* one whose condition is satisfied in ω (assuming for the moment that ω is complete enough to determine the truth values of the conditions). If no clause's condition is satisfied, or if the basic RV refers to objects that do not exist in ω, then the value is set by default to false for Boolean functions, null for other functions, and zero for number variables. If the condition in a clause is just "`true`", then the whole string "`if true then`" may be omitted.

In the applicable clause, each CPD argument is evaluated in ω. The resulting values are then passed to the elementary CPD. In the simplest case, the arguments are terms or formulas of $\mathcal{L}_M$, such as $\mathsf{State}(a, \mathsf{Pred}(t))$. An argument can also be a *set expression* of the form $\{\tau\ y : \varphi\}$, where τ is a type, y is a logical variable, and φ is a formula. The value of such an expression is the set of objects $o \in [\tau]^\omega$ such that ω satisfies φ with y bound to o. If the formula φ is just true it can be omitted: this is the case on line 9 of Fig. 1.

We require that the elementary CPDs obey two rules related to non-guaranteed objects. First, a CPD should never assign positive probability to objects that do not exist in the partially completed world ω. Thus, we allow an elementary CPD to assign positive probability to a non-guaranteed object only if the object was passed in as a CPD argument. Second, an elementary CPD cannot "peek" at the tuple representations of objects that are passed in: it must be invariant to permutations of the non-guaranteed objects.

4.2 Declarative semantics

So far we have explained BLOG semantics procedurally, in terms of a generative process. To facilitate both knowledge engineering and the development of learning algorithms, we would like to have declarative semantics. The standard approach — which is used in most existing FOPLs — is to say that a BLOG model defines a certain Bayesian network (BN) over the basic RVs. In this section we discuss how that approach needs to be modified for BLOG.

We will write σ to denote an instantiation of a set of RVs $\mathrm{vars}(\sigma)$, and σ_X to denote the value that σ assigns to X. If a BN is finite, then the probability it assigns to each complete instantiation σ is $P(\sigma) = \prod_{X \in \mathrm{vars}(\sigma)} p_X(\sigma_X | \sigma_{\mathrm{Pa}(X)})$, where p_X is the CPD for X and $\sigma_{\mathrm{Pa}(X)}$ is σ restricted to the parents of X. In an infinite BN, we can write a similar expression for each *finite* instantiation σ that is closed under the parent relation (that is, $X \in \mathrm{vars}(\sigma)$ implies $\mathrm{Pa}(X) \subseteq \mathrm{vars}(\sigma)$). If the BN is acyclic and each variable has finitely many ancestors, then these probability assignments define a unique distribution [Kersting and De Raedt, 2001].

The difficulty is that in the BN corresponding to a BLOG model, variables often have infinite parent sets. For instance, the BN for Ex. 1 has an infinite number of basic RVs of the form $V_{\mathsf{TrueColor}}[b]$: if it had only a finite number N of these RVs, it could not represent outcomes with more than N balls. Furthermore, each of these $V_{\mathsf{TrueColor}}[b]$ RVs is a parent of each $V_{\mathsf{ObsColor}}[d]$ RV, since if $\mathsf{BallDrawn}(d)$ happens to be b, then the observed color on draw d depends directly on the color of ball b. So the $V_{\mathsf{ObsColor}}[d]$ nodes have infinitely many parents. In such a model, assigning probabilities to finite instantiations that are closed under the parent relation does not define a unique distribution: in particular, it tells us nothing about the $V_{\mathsf{ObsColor}}[d]$ variables.

We required instantiations to be closed under the parent relation so that the factors $p_X(\sigma_X | \sigma_{\mathrm{Pa}(X)})$ would be well-defined. But we may not need the values of *all* of X's parents in order to determine the conditional distribution for X. For instance, knowing $V_{\mathsf{BallDrawn}}[d] = (\mathsf{Ball}, 13)$ and $V_{\mathsf{TrueColor}}[(\mathsf{Ball}, 13)] = \mathsf{Blue}$ is sufficient to determine the distribution for $V_{\mathsf{ObsColor}}[d]$: the colors of all the other balls are irrelevant in this context. We can read off this context-specific independence from the dependency statement for $\mathsf{ObsColor}$ in Fig. 1 by noting that the instantiation $(V_{\mathsf{BallDrawn}}[d] = (\mathsf{Ball}, 13), V_{\mathsf{TrueColor}}[(\mathsf{Ball}, 13)] = \mathsf{Blue})$ determines the value of the sole CPD argument $\mathsf{TrueColor}(\mathsf{BallDrawn}(d))$. We say this instantiation *supports* the variable $V_{\mathsf{ObsColor}}[d]$ (see [Milch *et al.*, 2005]).

Definition 5. *An instantiation σ* supports *a basic RV V of the form $V_f[o_1, \ldots, o_k]$ or $N_p[o_1, \ldots, o_k]$ if all possible worlds consistent with σ agree on (1) whether all the objects $o_1, \ldots, o_k$ exist, and, if so, on (2) the applicable clause in the dependency or number statement for V and the values for the CPD arguments in that clause.*

Note that some RVs, such as $N_{\mathsf{Ball}}[]$ in Ex. 1, are supported by the empty instantiation. We can now generalize the notion of being closed under the parent relation.

Definition 6. *A finite instantiation σ is* self-supporting *if its instantiated variables can be numbered $X_1, \ldots, X_N$ such that for each $n \leq N$, the restriction of σ to $\{X_1, \ldots, X_{n-1}\}$ supports X_n.*

This definition lets us give semantics to BLOG models in a way that is meaningful even when the corresponding BNs contain infinite parent sets. We will write $p_V(v \mid \sigma)$ for the probability that V's dependency or number statement assigns to the value v, given an instantiation σ that supports V.

Definition 7. *A distribution P over Ω_M* satisfies *a BLOG model M if for every finite, self-supporting instantiation σ with $\mathrm{vars}(\sigma) \subseteq \mathbf{V}_M$:*

$$P(\Omega_\sigma) = \prod_{n=1}^{N} p_{X_n}(\sigma_{X_n} \mid \sigma_{\{X_1, \ldots, X_{n-1}\}})$$

where Ω_σ is the set of possible worlds consistent with σ and $X_1, \ldots, X_N$ is a numbering of σ as in Def. 6.

A BLOG model is *well-defined* if there is exactly one probability distribution that satisfies it. Recall that a BN is well-defined if it is acyclic and each variable has a finite set of ancestors. Another way of saying this is that each variable can be "reached" by enumerating its ancestors in a finite, topologically ordered list. The well-definedness criterion for BLOG is similar, but deals with finite, self-supporting instantiations rather than finite, topologically ordered lists of variables.

Theorem 1. *Let M be a BLOG model. Suppose that $\mathbf{V}_M$ is at most countably infinite,*[1] *and for each $V \in \mathbf{V}_M$ and $\omega \in \Omega_M$, there is a self-supporting instantiation that agrees with ω and includes V. Then M is well-defined.*

The theorem follows from a result in [Milch *et al.*, 2005] that deals with distributions over full instantiations of a set of RVs. Prop. 1 makes the connection between full instantiations of $\mathbf{V}_M$ and possible worlds.

To check that the criterion of Thm. 1 holds for a particular example, we need to consider each basic RV. In Ex. 1, the number RV for balls is supported by the empty instantiation, so in every world it is part of a self-supporting instantiation of size one. Each $\mathsf{TrueColor}(b)$ RV depends only on whether its argument exists, so these variables participate in self-supporting instantiations of size two. Similarly, each $\mathsf{BallDrawn}$ variable depends only on what balls exist. To sample an $\mathsf{ObsColor}(d)$ variable, we need to know $\mathsf{BallDrawn}(d)$ and $\mathsf{TrueColor}(\mathsf{BallDrawn}(d))$, so these variables are in self-supporting instantiations of size four. Similar arguments can be made for Examples 2 and 3. Of course, we would like to have an algorithm for checking whether a BLOG model is well-defined; the criteria given in Thm. 2 in Sec. 6.2 are a first step in this direction.

5 Evidence and Queries

Because a well-defined BLOG model M defines a distribution over model structures, we can use arbitrary sentences of $\mathcal{L}_M$ as evidence and queries. But sometimes such sentences are not enough. In Ex. 3, the user observes radar blips, which are not referred to by any terms in the language. The user could assert evidence about the blips using existential quantifiers, but then how could he make a query of the form, "Did *this* blip come from the same aircraft as *that* blip?"

A natural solution is to allow the user to extend the language when evidence arrives, adding constant symbols to refer to observed objects. In many cases, the user observes some new objects, introduces some new symbols, and assigns the symbols to the objects in an uninformative order. To handle such cases, BLOG includes a special macro. For instance, given 4 radar blips at time 8, one can assert:

```
{Blip r: Time(r) = 8} = {Blip1, Blip2, Blip3, Blip4};
```

[1]This is satisfied if the Real and $\mathsf{R}k\mathsf{Vector}$ types are not arguments to random functions or return types of generating functions.

This introduces the new constants $\mathsf{Blip1}, \ldots, \mathsf{Blip4}$ and asserts that there are exactly 4 radar blips at time 8.

Formally, the macro augments the model with dependency statements for the new symbols. The statements implement sampling without replacement; for our example, we have

```
Blip1 ~ Uniform({Blip r : (Time(r) = 8)});
Blip2 ~ Uniform({Blip r : (Time(r) = 8) & (Blip1 != r)});
```

and so on. Once the model has been extended this way, the user can make assertions about the apparent positions of Blip1, Blip2, etc., and then use these symbols in queries.

These new constants resemble Skolem constants, but conditioning on assertions about the new constants is *not* the same as conditioning on an existential sentence. For example, suppose you go into a new wine shop, pick up a bottle at random, and observe that it costs \$40. This scenario is correctly modeled by introducing a new constant Bottle1 with a `Uniform` CPD. Then observing that Bottle1 costs over \$40 suggests that this is a fancy wine shop. On the other hand, the mere *existence* of a \$40+ bottle does not suggest this, because almost every shop has *some* bottle at over \$40.

6 Inference

Because the set of basic RVs of a BLOG model can be infinite, it is not obvious that inference for well-defined BLOG models is even decidable. However, the generative process intuition suggests a rejection sampling algorithm. We present this algorithm not because it is particularly efficient, but because it demonstrates the decidability of inference for a large class of BLOG models (see Thm. 2 below) and illustrates several issues that any BLOG inference algorithm must deal with. At the end of this section, we present experimental results from a somewhat more efficient likelihood weighting algorithm.

6.1 Rejection sampling

Suppose we are given a partial instantiation e as evidence, and a query variable Q. Our rejection sampling algorithm starts by imposing an arbitrary numbering on the the basic RVs. To generate each sample, it starts with an empty instantiation σ. Then it repeats the following process: scan the basic RVs in the imposed order until we reach the first RV V that is supported by σ but not already instantiated in σ; sample a value v for V according to V's dependency statement; and augment σ with the assignment $V = v$. It continues until all the query and evidence variables have been sampled. If the sample is consistent with the evidence e, then it increments a counter N_q, where q is the sampled value of Q. Otherwise, it rejects this sample. After N accepted samples, the estimate of $P(Q = q \mid e)$ is N_q/N.

This algorithm requires a subroutine that determines whether a partial instantiation σ supports a basic RV V, and if so, returns a sample from V's conditional distribution. For a basic RV V of the form $V_f\,[o_1, \ldots, o_k]$ or $N_p\,[o_1, \ldots, o_k]$, the subroutine begins by checking the values of the relevant number variables in σ to determine whether all of $o_1, \ldots, o_k$ exist. If some of these number variables are not instantiated, then σ does not support V. If some of $o_1, \ldots, o_k$ do not exist, the subroutine returns the default value for V. If they do all exist, the subroutine follows the semantics for dependency statements discussed in Sec. 4.1. First, it iterates over the clauses in the dependency (or number) statement until it reaches a clause whose condition is either undetermined or determined to be true given σ (if all the conditions are determined to be false, then it returns the default value for V). If the condition is undetermined, then σ does not support V. If it is determined to be true, then the subroutine evaluates each of the CPD arguments in this clause. If σ determines the values of all the arguments, then the subroutine samples a value for V by passing those values to the `sample` method of this clause's elementary CPD. Otherwise, σ does not support V.

To evaluate terms and quantifier-free formulas, we use a straightforward recursive algorithm. The base case looks up the value of a particular function application RV in σ and returns "undetermined" if it is not instantiated. A formula may be determined even if some of its subformulas are not determined: for example, $\alpha \wedge \beta$ is determined to be false if α is false. It is more complicated to evaluate set expressions such as `{Blip r: Time(r) = 8}`, which can be used as CPD arguments. A naive algorithm for evaluating this expression would first enumerate all the objects of type Blip (which requires certain number variables to be instantiated), then select the blips r that satisfy $\mathsf{Time}(r) = 8$. But Fig. 3 specifies that there may exist some blips for each aircraft a and each natural number t: since there are infinitely many natural numbers, some worlds contain infinitely many blips. Fortunately, the number of blips r with $\mathsf{Time}(r) = 8$ is necessarily finite: in every world there are a finite number of aircraft, and each one generates a finite number of blips at time 8. We have an algorithm that scans the formula within a set expression for *generating function restrictions* such as $\mathsf{Time}(r) = 8$, and uses them to avoid enumerating infinite sets when possible. A similar method is used for evaluating quantified formulas.

6.2 Termination criteria

In order to generate each sample, the algorithm above repeatedly instantiates the first variable that is supported but not yet instantiated, until it instantiates all the query and evidence variables. When can we be sure that this will take a finite amount of time? The first way this process could fail to terminate is if it goes into an infinite loop while checking whether a particular variable is supported. This happens if the program ends up enumerating an infinite set while evaluating a set expression or quantified formula. We can avoid this by ensuring that all such expressions in the BLOG model are finite once generating function restrictions are taken into account.

The sample generator also fails to terminate if it never constructs an instantiation that supports a particular query or evidence variable. To see how this can happen, consider calling the subroutine described above to sample a variable V. If V is not supported, the subroutine will realize this when it encounters a variable U that is relevant but not instantiated. Now consider a graph over basic variables where we draw an edge from U to V when the evaluation process for V hits U in this way. If a variable is never supported, then it must be part of a cycle in this graph, or part of a receding chain of variables $V_1 \leftarrow V_2 \leftarrow \cdots$ that is extended infinitely.

The graph constructed in this way varies from sample to sample: for instance, sometimes the evaluation process for $V_{\mathsf{ObsColor}}\,[d]$ will hit $V_{\mathsf{TrueColor}}\,[(\mathsf{Ball}, 7)]$, and sometimes it will hit $V_{\mathsf{TrueColor}}\,[(\mathsf{Ball}, 13)]$. However, we can rule out cycles and infinite receding chains in all these graphs by considering a more abstract graph over function symbols and types.

Definition 8. *The* symbol graph *for a* BLOG *model M is a directed graph whose nodes are the types and random function symbols of M, where the parents of a type τ or function symbol f are:*

- *the random function symbols that occur on the right hand side of the dependency statement for f or some number statement for τ;*
- *the types of variables that are quantified over in formulas or set expressions on the right hand side of such a statement;*
- *the types of the arguments for f or the return types of generating functions for τ.*

If the sampling subroutine for a basic RV V hits a basic RV U, then there must be an edge from U's function symbol (or type, if U is a number RV) to V's function symbol (or type) in the symbol graph. This property, along with ideas from [Milch *et al.*, 2005], allows us to prove the following:

Theorem 2. *Suppose M is a* BLOG *model where:*

1. *uncountable built-in types do not serve as function arguments or as the return types of generating functions;*
2. *each quantified formula and set expression ranges over a finite set once generating function restrictions are taken into account;*
3. *the symbol graph is acyclic.*

Then M is well-defined. Also, for any evidence instantiation e and query variable Q, the rejection sampling algorithm described in Sec. 6.1 converges to the posterior $P(Q|e)$ defined by the model, taking finite time per sampling step.

The criteria in Thm. 2 are very conservative: in particular, when we construct the symbol graph, we ignore all structure in the dependency statements and just check for the occurrence of function and type symbols. These criteria are satisfied by the models in Figures 1 and 2. However, the model in Fig. 3 does not satisfy the criteria because there is a self-loop from State to State. The criteria do not exploit the fact that $\mathsf{State}(a, t)$ depends only on $\mathsf{State}(a, \mathsf{Pred}(t))$, and the nonrandom function Pred is acyclic. Friedman *et al.* [1999] have already dealt with this issue in the context of probabilistic relational models; their algorithm can be adapted to obtain a stronger version of Thm. 2 that covers Fig. 3.

6.3 Experimental results

Milch *et al.* [2005] describe a guided likelihood weighting algorithm that uses backward chaining from the query and evidence nodes to avoid sampling irrelevant variables. This algorithm can also be adapted to BLOG models. We applied this algorithm for Ex. 1, asserting that 10 balls were drawn and all appeared blue, and querying the number of balls in the urn. The top graph of Fig. 4 shows that when the prior for the number of balls is uniform over $\{1, \ldots, 8\}$, the posterior puts more weight on small numbers of balls; this makes sense because the more balls there are in the urn, the less likely it is that they are all blue. The bottom graph, using a Poisson(6) prior, shows a similar but less pronounced effect. Note that the posterior probabilities computed by the likelihood weighting algorithm are very close to the exact values (computed by exhaustive enumeration). These results could not be obtained using any algorithm that constructed a single fixed BN, since the number of potentially relevant $V_{\mathsf{TrueColor}}[b]$ variables is infinite in the Poisson case.

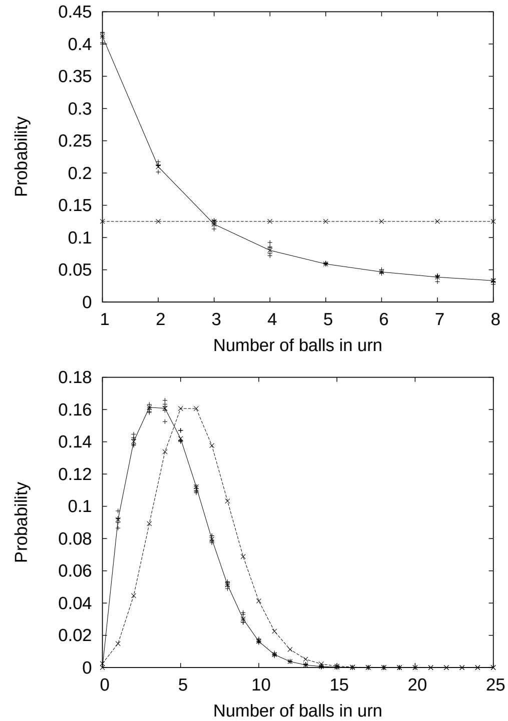

Figure 4: Distribution for the number of balls in the urn (Ex. 1). Dashed lines are the uniform prior (top) or Poisson prior (bottom); solid lines are the exact posterior given that 10 balls were drawn and all appeared blue; and plus signs are posterior probabilities computed by 5 independent runs of 20,000 samples (top) or 100,000 samples (bottom).

7 Related Work

Gaifman [1964] was the first to suggest defining a probability distribution over first-order model structures. Halpern [1990] defines a language in which one can make statements about such distributions: for instance, that the probability of the set of worlds that satisfy $\mathsf{Flies}(\mathsf{Tweety})$ is 0.8. *Probabilistic logic programming* [Ng and Subrahmanian, 1992] can be seen as an application of this approach to Horn-clause knowledge bases. Such an approach only defines *constraints* on distributions, rather than defining a unique distribution.

Most first-order probabilistic languages (FOPLs) that define unique distributions fix the set of objects and the interpretations of (non-Boolean) function symbols. Examples include relational Bayesian networks [Jaeger, 2001] and Markov logic models [Domingos and Richardson, 2004]. Prolog-based languages such as probabilistic Horn abduction [Poole, 1993], PRISM [Sato and Kameya, 2001], and Bayesian logic programs [Kersting and De Raedt, 2001] work with *Herbrand models*, where the objects are in one-to-one correspondence with the ground terms of the language.

There are a few FOPLs that allow explicit *reference uncertainty*, i.e., uncertainty about the interpretations of function symbols. Among these are two languages that use indexed RVs rather than logical notation: BUGS [Gilks *et al.*, 1994] and indexed probability diagrams (IPDs) [Mjolsness, 2004]. Reference uncertainty can also be represented in probabilistic relational models (PRMs) [Koller and Pfeffer, 1998], where a “single-valued complex slot” corresponds to an uncertain unary function. PRMs are unfortunately restricted to unary functions (attributes) and binary predicates (relations). Probabilistic entity-relationship models [Heckerman *et al.*, 2004] lift this restriction, but represent reference uncertainty using relations (such as $\mathsf{Drawn}(d, b)$) and special mutual exclusivity constraints, rather than with functions such as $\mathsf{BallDrawn}(d)$. Multi-entity Bayesian network logic (MEBN) [Laskey, 2004] is similar to BLOG in allowing uncertainty about the values of functions with any number of arguments.

The need to handle unknown objects has been appreciated since the early days of FOPL research: Charniak and Goldman’s plan recognition networks (PRNs) [1993] can contain unbounded numbers of objects representing hypothesized plans. However, external rules are used to decide what objects and variables to include in a PRN. While each possible PRN defines a distribution on its own, Charniak and Goldman do not claim that the various PRNs are all approximations to some single distribution over outcomes.

Some more recent FOPLs do define a single distribution over outcomes with varying objects. IPDs allow uncertainty over the index range for an indexed family of RVs. PRMs and their extensions allow a variety of forms of uncertainty about the number (or existence) of objects satisfying certain relational constraints [Koller and Pfeffer, 1998; Getoor *et al.*, 2001] or belonging to each type [Pasula *et al.*, 2003]. However, there is no unified syntax or semantics for dealing with unknown objects in PRMs. MEBNs take yet another approach: an MEBN model includes a set of unique identifiers, for each of which there is an “identity” RV indicating whether the named object exists.

Our approach to unknown objects in BLOG can be seen as unifying the PRM and MEBN approaches. Number statements neatly generalize the various ways of handling unknown objects in PRMs: number uncertainty [Koller and Pfeffer, 1998] corresponds to a number statement with a single generating function; existence uncertainty [Getoor *et al.*, 2001] can be modeled with two or more generating functions (and a CPD whose support is $\{0, 1\}$); and domain uncertainty [Pasula *et al.*, 2003] corresponds to a number statement with no generating functions. There is also a correspondence between BLOG and MEBN logic: the tuple representations in a BLOG model can be thought of as unique identifiers in an MEBN model. The difference is that BLOG determines which objects actually exist in a world using number variables rather than individual existence variables.

Finally, it is informative to compare BLOG with the IBAL language [Pfeffer, 2001], in which a program defines a distribution over outputs that can be arbitrary nested data structures. An IBAL program could implement a BLOG-like generative process with the outputs viewed as logical model structures. But the declarative semantics of such a program would be less clear than the corresponding BLOG model.

8 Conclusion

BLOG is a representation language for probabilistic models with unknown objects. It contributes to the solution of a very general problem in AI: intelligent systems must represent and reason about objects, but those objects may not be known *a priori* and may not be directly and uniquely identified by perceptual processes. Our approach defines generative models that create first-order model structures by adding objects and setting function values; everything else follows naturally from this design decision. Much remains to be done, especially on inference: we expect to employ MCMC with user-defined and possibly adaptive proposal distributions, and to develop algorithms that work directly with objects rather than at the lower level of basic RVs.

References

[Charniak and Goldman, 1993] E. Charniak and R. P. Goldman. A Bayesian model of plan recognition. *AIJ*, 64(1):53–79, 1993.

[Domingos and Richardson, 2004] P. Domingos and M. Richardson. Markov logic: A unifying framework for statistical relational learning. In *Proc. ICML Wksp on Statistical Relational Learning and Its Connections to Other Fields*, 2004.

[Friedman *et al.*, 1999] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Proc. 16th IJCAI*, pages 1300–1307, 1999.

[Gaifman, 1964] H. Gaifman. Concerning measures in first order calculi. *Israel J. Math.*, 2:1–18, 1964.

[Getoor *et al.*, 2001] L. Getoor, N. Friedman, D. Koller, and B. Taskar. Learning probabilistic models of relational structure. In *Proc. 18th ICML*, pages 170–177, 2001.

[Gilks *et al.*, 1994] W. R. Gilks, A. Thomas, and D. J. Spiegelhalter. A language and program for complex Bayesian modelling. *The Statistician*, 43(1):169–177, 1994.

[Halpern, 1990] J. Y. Halpern. An analysis of first-order logics of probability. *AIJ*, 46:311–350, 1990.

[Heckerman *et al.*, 2004] D. Heckerman, C. Meek, and D. Koller. Probabilistic models for relational data. Technical Report MSR-TR-2004-30, Microsoft Research, 2004.

[Jaeger, 2001] M. Jaeger. Complex probabilistic modeling with recursive relational Bayesian networks. *Annals of Math and AI*, 32:179–220, 2001.

[Kersting and De Raedt, 2001] K. Kersting and L. De Raedt. Adaptive Bayesian logic programs. In *11th Int. Conf. on ILP*, 2001.

[Koller and Pfeffer, 1998] D. Koller and A. Pfeffer. Probabilistic frame-based systems. In *Proc. 15th AAAI*, pages 580–587, 1998.

[Laskey, 2004] K. B. Laskey. MEBN: A logic for open-world probabilistic reasoning. Technical report, George Mason Univ., 2004.

[Milch *et al.*, 2005] B. Milch, B. Marthi, D. Sontag, S. Russell, D. L. Ong, and A. Kolobov. Approximate inference for infinite contingent Bayesian networks. In *10th AISTATS Wksp*, 2005.

[Mjolsness, 2004] E. Mjolsness. Labeled graph notations for graphical models. Technical Report 04-03, School of Information and Computer Science, UC Irvine, 2004.

[Ng and Subrahmanian, 1992] R. T. Ng and V. S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, 1992.

[Pasula *et al.*, 2003] H. Pasula, B. Marthi, B. Milch, S. Russell, and I. Shpitser. Identity uncertainty and citation matching. In *NIPS 15*. MIT Press, Cambridge, MA, 2003.

[Pearl, 1988] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, San Francisco, revised edition, 1988.

[Pfeffer, 2001] A. Pfeffer. IBAL: A probabilistic rational programming language. In *Proc. 17th IJCAI*, pages 733–740, 2001.

[Poole, 1993] D. Poole. Probabilistic Horn abduction and Bayesian networks. *AIJ*, 64(1):81–129, 1993.

[Sato and Kameya, 2001] T. Sato and Y. Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *JAIR*, 15:391–454, 2001.

[Sittler, 1964] R. W. Sittler. An optimal data association problem in surveillance theory. *IEEE Trans. Military Electronics*, MIL-8:125–139, 1964.