

Semantics and Inference for Recursive Probability Models

Avi Pfeffer

Division of Engineering and Applied Sciences
Harvard University
Cambridge, MA 02138
avi@eecs.harvard.edu

Daphne Koller

Computer Science Department
Stanford University
Stanford, CA 94305
koller@cs.stanford.edu

Abstract

In recent years, there have been several proposals that extend the expressive power of Bayesian networks with that of relational models. These languages open the possibility for the specification of recursive probability models, where a variable might depend on a potentially infinite (but finitely describable) set of variables. These models are very natural in a variety of applications, e.g., in temporal, genetic, or language models. In this paper, we provide a structured representation language that allows us to specify such models, a clean measure-theoretic semantics for this language, and a probabilistic inference algorithm that exploits the structure of the language for efficient query-answering.

1 Introduction

There has been a growing interest in recent years in *relational probabilistic languages* (Wellman, Breese, & Goldman 1992; Ngo & Haddawy 1996; Koller & Pfeffer 1998). These languages combine the ability of Bayesian networks to compactly describe probability models with the generality, flexibility and modularity of logical representations. They have extended the applicability of probabilistic reasoning techniques to more complex domains than in the past.

In the relational setting, it is very natural to define probability models that are *recursive*. In a recursive model, a variable associated with a particular domain entity can depend probabilistically on the same variable associated with a different entity. Examples of recursive probability models are temporal models, genetic models of gene propagation, and stochastic grammar models of natural language. Recursive models are challenging, because in principle they describe distributions over infinitely many variables.

Existing languages do not deal adequately with recursive probability models. Some languages, e.g. (Koller & Pfeffer 1998), rule them out altogether. Others, such as (Ngo & Haddawy 1996), get around the issue by using a variant of the closed world assumption to limit the set of variables. This assumption is often inappropriate, since it requires that all entities and the relationships between them be known and stated explicitly. The language of (Koller, McAllester, & Pfeffer 1997) allows the specification of recursive probability

models as stochastic computational processes, but its semantics is quite limiting, since it requires that a process is guaranteed to terminate in order for the model to be well-defined.

In this paper, we present a new framework for recursive probability models. We do not make any restrictive assumptions on the language, explicitly considering probability distributions over infinitely many variables. We provide a natural measure-theoretic semantics for models in our language, that extends the semantics of Bayesian networks in a natural way. We also present a general anytime approximate inference algorithm for our language. Our algorithm exploits the relational structure of the domain, and we show that doing so yields great benefits for anytime inference.

2 Recursive Probability Models

For ease of presentation, we will focus on a simplified version of our full language. We consider the impact of recursion on more expressive languages in Section 5.

The main unit of discourse in our language is a *class*, which describes a particular kind of object, with a probability model over its properties. A class has *simple* and *complex attributes*. Intuitively, a simple attribute describes a basic property of an object, while a complex attribute indicates a relationship between one object and another. A simple attribute has an associated probability model, describing how the value of the attribute depends on other attributes of the same object, and on attributes of related objects.

Definition 2.1: A *probabilistic relational knowledge base* consists of:

- A set \mathcal{C} of *classes*.
- A set \mathcal{A} of *complex attributes*. Each complex attribute A has a *domain type* $Dom[A] \in \mathcal{C}$ and a *range type* $Range[A] \in \mathcal{C}$.
- A set \mathcal{B} of *simple attributes*. Each simple attribute B has a *domain type* $Dom[A] \in \mathcal{C}$, and a *range*, which is a finite set of values, denoted by $Val[B]$. The simple attribute B has an associated probability model, consisting of
 - A set $Pa[B]$ of *parents*, each of which is a *well-typed terminal attribute chain* (see below) $\sigma = A_1 \dots A_n$, such that $Dom[A_1] = Dom[B]$. The *range* of σ is the range of A_n , and is denoted by $Val[\sigma]$.
 - A *conditional probability function* CPF_B , defining a probability distribution over $Val[B]$ for each assign-

ment of values to $Pa[B]$. Formally, if B has parents $\sigma_1, \dots, \sigma_m$, $CPF_B(w | \mathbf{v})$ defines the conditional probability over values $w \in Val[B]$ given an assignment $\mathbf{v} = (v_1, \dots, v_m)$ to $\sigma_1, \dots, \sigma_m$.

- A single instance I_T , known as the *top-level instance*, whose *type* is a class in \mathcal{C} .

An attribute chain $\sigma = A_1 \dots A_n$ is *well-typed* if for each $i = 1, \dots, n - 1$, A_i is complex, and $Dom[A_{i+1}] = Range[A_i]$. It is *terminal* if A_n is simple. ■

As in Bayesian networks, one must make sure that the probability model is acyclic. For our simple language, it turns out that it is sufficient to make sure that there is no class C and sequence of simple attributes A_1, \dots, A_n whose domain type is C , such that A_{i+1} is a parent of A_i and A_1 is a parent of A_n .

Example 2.2: As a running example we will consider a simple genetic model for the transmission and manifestation of a single eye-color gene. The knowledge base \mathcal{K} consists of a single class **Person**, complex attributes **Mother** and **Father**, and simple attributes **M-Chromosome**, **P-Chromosome** and **Phenotype**. The domain and range types of each complex attribute are **Person**. The domain type of each simple attribute is **Person**, while its range is $\{pink, mauve\}$. The parents of **Phenotype** are **M-Chromosome** and **P-Chromosome**, and its CPF specifies that if either parent is *pink*, **Phenotype** is *pink* with probability 0.99, otherwise it is *mauve* with probability 0.99. The parents of **M-Chromosome** are **Mother.M-Chromosome**, and **Mother.P-Chromosome**, while **P-Chromosome** depends on the father's chromosomes. \mathcal{K} also contains the named instance *Fred* of type **Person**. ■

This simple example already shows how easy and natural it is to create infinitely recursive models in this language. A possible world for our language consists of all entities related to the top-level instance via some finite attribute chain. We make the simplifying assumption here that no entity can be reached by two distinct attribute chains. The set of entities in the world is therefore fixed (but possibly infinite), and is in one to one correspondence with the set of finite complex attribute chains. For our example, it consists of distinct entities for *Fred* and all his ancestors. Since a possible world is characterized by the values of the simple attributes of the domain entities, and the set of domain entities is fixed, we can define a possible world simply as follows.

Definition 2.3: Let \mathcal{K} be a probabilistic relational KB. A *variable* of \mathcal{K} has the form $I_T.\sigma$, where $\sigma = A_1 \dots A_n$ is a well-typed terminal attribute chain such that $Dom[A_1]$ is the type of I_T . We will denote variables by the letters X, Y and Z . A *possible world* ω for \mathcal{K} is a function that maps each variable X of \mathcal{K} to an element in $Val[X]$. The set of possible worlds for \mathcal{K} is denoted $\Omega_{\mathcal{K}}$, or simply Ω . ■

3 Measure-Theoretic Semantics

Intuitively, a probabilistic relational KB defines an infinite Bayesian network. Each variable $X = I_T.A_1 \dots A_n$ has a set of parents and a conditional probability function, as

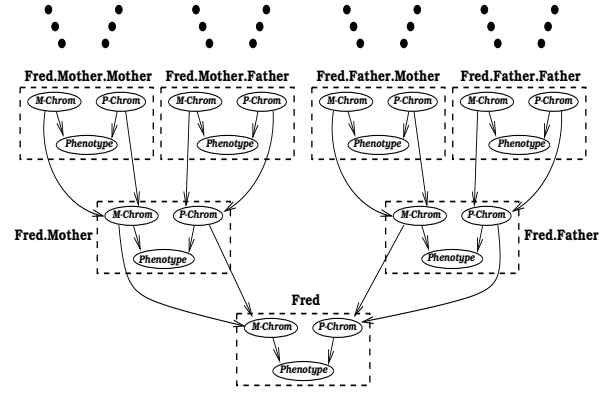


Figure 1: Infinite BN for genetic example.

specified by the probability model of A_n . Figure 1 illustrates the infinite BN for the KB of Example 2.2. To emphasize the object structure of the model, all variables associated with a particular domain entity are enclosed in a box.

Since a probabilistic relational KB defines an infinite BN, it may seem natural to define the semantics of our language in the same way as for BNs, namely, that the probability of any possible world ω is given by $\prod_X CPF_X(\omega(X) | \omega(Pa[X]))$. Unfortunately, if there are infinitely many variables, the probability of any possible world is typically 0.

Instead, we need to define a *probability measure* over the possible worlds. A probability measure assigns a probability to sets of possible worlds rather than individual worlds. Not all sets of worlds are assigned a probability, only those in a particular set \mathcal{E} of subsets of Ω . A set $E \in \mathcal{E}$ is called an *event*, and \mathcal{E} is called the *event space*. The set \mathcal{E} is required to be a σ -algebra, which means that it is closed under countable unions and complements. A probability measure μ over Ω, \mathcal{E} is a function from \mathcal{E} to $[0, 1]$ such that $\mu(\Omega) = 1$, and that is *countably additive*, i.e., if $E \in \mathcal{E}$ is the finite or countable union of disjoint events F_i , $\mu(E) = \sum_i \mu(F_i)$.

Definition 3.1: Let \mathcal{K} be a probabilistic relational knowledge base. A *basic event* of \mathcal{K} has the form $[\mathbf{X} = \mathbf{x}]$, where \mathbf{X} is a finite set of variables of \mathcal{K} , and \mathbf{x} is an assignment of values to \mathbf{X} . The basic event $[\mathbf{X} = \mathbf{x}]$ denotes the set of possible worlds ω such that $\omega(X_i) = x_i$ for each $X_i \in \mathbf{X}$. The set of basic events of \mathcal{K} will be denoted $\mathcal{F}_{\mathcal{K}}$. We define $\mathcal{E}_{\mathcal{K}}$ to be the set of finite or countable unions of sets in $\mathcal{F}_{\mathcal{K}}$. ■

It is easy to verify that $\mathcal{E}_{\mathcal{K}}$ is a σ -algebra, and it will serve as our event space. We now proceed to define the semantics of our language in a natural way. A model of \mathcal{K} is a probability measure μ over $\Omega_{\mathcal{K}}, \mathcal{E}_{\mathcal{K}}$ that respects the probabilistic knowledge in \mathcal{K} . Intuitively, we require that μ locally satisfy the structure of the infinite BN. If we look at a finite set \mathbf{X} of variables, we can consider the fragment of the infinite BN containing \mathbf{X} , and require that the distribution over \mathbf{X} induced by μ agree with the BN fragment.

Definition 3.2: Let \mathbf{X} be a finite set of variables. We define the *roots* of \mathbf{X} to be $\cup_{X \in \mathbf{X}} Pa[X] - \mathbf{X}$. Let \mathbf{Y} denote the roots of \mathbf{X} . The *BN fragment over \mathbf{X}* , written $B_{\mathbf{X}}$, is a DAG over $\mathbf{X} \cup \mathbf{Y}$, in which there is an edge from X_1 to X_2

if $X_2 \in \mathbf{X}$ and $X_1 \in Pa[X_2]$. $B_{\mathbf{X}}$ defines a conditional probability distribution over \mathbf{X} given \mathbf{Y} , by

$$B_{\mathbf{X}}(\mathbf{X} = \mathbf{x} \mid \mathbf{Y} = \mathbf{y}) = \prod_X CPF_X(X = x \mid Pa[X] = \mathbf{u}_X),$$

where \mathbf{u}_X denotes the value assigned to $Pa[X]$ in \mathbf{x}, \mathbf{y} .

If φ is some distribution over \mathbf{Y} , the notation $\varphi \cdot B_{\mathbf{X}}$ denotes the probability distribution π over \mathbf{X} defined by $\pi(\mathbf{x}) = \sum_{\mathbf{y}} \varphi(\mathbf{y}) B_{\mathbf{X}}(\mathbf{x} \mid \mathbf{y})$. ■

In order for the BN fragment over a set of variables to be meaningful, it must capture all the probabilistic relationships between variables in the set. If X and Y are variables in some set \mathbf{X} , and there is a path of influences from X to Y that passes through some variable Z not in \mathbf{X} , then we cannot expect the BN fragment over \mathbf{X} to correctly express the conditional probability of Y given X . We will require that a model of a KB agree with the BN fragments over sets of variables that are *self-contained* in the following sense.

Definition 3.3: A set of variables \mathbf{X} is *self-contained* if for all directed paths $X_1 \rightarrow \dots \rightarrow X_n$ such that X_1 and X_n are in \mathbf{X} , all X_i are in \mathbf{X} . ■

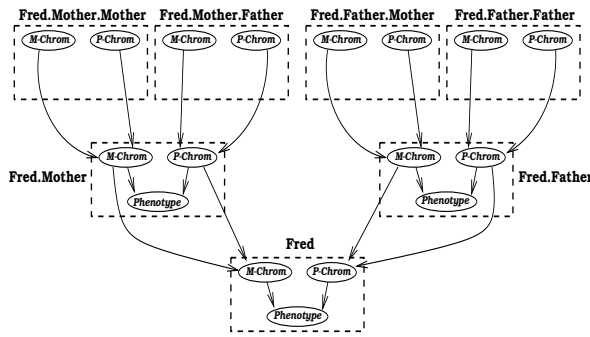


Figure 2: BN fragment for genetic example.

In our example, the variables associated with *Fred* and his parents form a self-contained set. The roots of this set are the chromosome variables of the grandparents. The BN fragment over this set of variables is shown in Figure 2. Essentially, the BN fragment over a self-contained set of variables captures all the knowledge expressed in the local probability models for the variables in the set. This includes both the conditional independence relationships in the structure of the BN fragment, and the quantitative knowledge in the CPFs. The following lemma shows how this knowledge is preserved when we move from one set of variables to a larger set.

Lemma 3.4 Let \mathbf{X}_1 and \mathbf{X}_2 be two finite self-contained sets of variables, with $\mathbf{X}_1 \subseteq \mathbf{X}_2$. Let \mathbf{Y}_1 and \mathbf{Y}_2 be the roots of \mathbf{X}_1 and \mathbf{X}_2 respectively. Then \mathbf{X}_1 is d -separated from \mathbf{Y}_2 by \mathbf{Y}_1 in $B_{\mathbf{X}_2}$.

Lemma 3.4 is illustrated in Figure 2. Let \mathbf{X}_1 be the set of variables associated with *Fred*, while \mathbf{X}_2 is the set of variables associated with *Fred* and his parents. \mathbf{Y}_1 is the set of chromosome variables of *Fred's* parents, while \mathbf{Y}_2 is the set

of chromosome variables of his grandparents. It is easy to see in the figure that \mathbf{Y}_1 d -separates \mathbf{X}_1 from \mathbf{Y}_2 . The condition that \mathbf{X}_1 be self-contained is crucial to the truth of the lemma.

We are now ready to define the semantics of recursive probability models in a very natural way. We simply require that a model of a KB respect the BN fragment over every finite self-contained set of variables of the KB.

Definition 3.5: Let \mathcal{K} be a recursive probabilistic KB. A *model* for \mathcal{K} is a probability measure μ over $(\Omega_{\mathcal{K}}, \mathcal{E}_{\mathcal{K}})$, such that for every finite self-contained set of variables \mathbf{X} of \mathcal{K} , with roots \mathbf{Y} , $\mu(\mathbf{X} \mid \mathbf{Y}) = B_{\mathbf{X}}(\mathbf{X} \mid \mathbf{Y})$. ■

For a KB that has only a finite set of variables, this definition reduces to the standard semantics of BNs. All the variables of the KB form a finite self-contained set, and the BN fragment over all the variables is just an ordinary BN over those variables. So our semantics is a natural generalization of standard BN semantics. One of the nice properties of finite BNs is that a BN defines a unique probability distribution. Unfortunately, this is no longer the case for recursive probability models.

Example 3.6 Consider a KB with a single class *State*, with simple attribute *Current* and complex attribute *Previous* of type *State*. The attribute *Current* has a single parent *Previous.Current*. This class defines a Markov chain “looking backwards”. If π is any stationary distribution of the Markov chain, the KB has a model whose marginal distribution over the state at any time is p . If the KB describes a non-ergodic Markov chain that has multiple stationary distributions, it has more than one model. ■

Interestingly, however, it turns out that every KB has at least one model.

Theorem 3.7: Let \mathcal{K} be a recursive probabilistic KB. There exists a probability measure μ over $(\Omega_{\mathcal{K}}, \mathcal{E}_{\mathcal{K}})$ that is a model for \mathcal{K} .

Sketch of proof (full proofs of all theorems can be found in (Pfeffer 2000)): We begin by defining a sequence of larger and larger self-contained sets of variables, that eventually covers all the variables of \mathcal{K} . We can achieve this by setting \mathbf{X}_i to be the set of variables whose chains have length $\leq i$. (Such sets are self-contained for our language.) Next, for $i \geq j$, we define S_j^i to be the set of probability distributions π over \mathbf{X}_j , such that there is some distribution φ over the roots of \mathbf{X}_i , such that π is the marginal of $\varphi \cdot B_{\mathbf{X}_i}$ over \mathbf{X}_j . We can show that S_j^i is closed and non-empty, and, using Lemma 3.4, that $S_j^{i+1} \subseteq S_j^i$. Therefore the sequence $(S_j^i)_{i=j}^{\infty}$ is a non-increasing sequence of closed, non-empty sets, so by compactness of the space of probability distributions over \mathbf{X}_j , it has a non-empty intersection, which we denote by S_j .

Next, we use the S_j to recursively construct a sequence of distributions over all the \mathbf{X}_j as follows. μ_1 is any distribution in S_1 , and for $j > 1$, μ_j is a distribution in S_j such that μ_{j-1} is the marginal of μ_j over \mathbf{X}_{j-1} . This construction is always possible by definition of S_j . We can

then define a function μ over the basic events $\mathcal{F}_{\mathcal{K}}$, by setting $\mu(E) = \mu_j(E)$ where all variables mentioned by E are in \mathbf{X}_j . By construction, it does not matter which j we pick, since all project to the same marginal distribution over the variables mentioned by E . We then show that μ so defined is countably additive, and that $\mu(\Omega_{\mathcal{K}}) = 1$, so we can extend μ to a probability measure over $\Omega_{\mathcal{K}}, \mathcal{E}_{\mathcal{K}}$. Finally, we show that μ so constructed does indeed satisfy the conditions of Definition 3.5. ■

Note the central role of compactness in the proof. The result is a compactness result, and resembles the compactness theorem for first-order logic (Enderton 1972). The key idea is that a KB specifies a set of *local* constraints on the probability model, and the proof shows that if it is possible to satisfy every finite set of local constraints, it is possible to satisfy all constraints simultaneously. In our case, every finite set of constraints can be captured within a BN fragment, so it can always be satisfied. It follows that every KB has a model.

4 Inference

The traditional approach to inference in relational probability models has been to use the technique of *knowledge based model construction (KBMC)* (Wellman, Breese, & Goldman 1992). In this approach, used in (Ngo & Haddawy 1996) and (Koller & Pfeffer 1998), a Bayesian network is constructed in order to solve a particular query on a knowledge base. The network consists of all nodes that are relevant to determining the probability of the query variables given the evidence. Obviously, this approach will not work for recursive probability models, since the set of relevant nodes may be infinite, in which case the BN construction process will not terminate. The approaches of (Koller, McAllester, & Pfeffer 1997) and (Pfeffer *et al.* 1999) suffer from the same shortcoming. Even though they do not explicitly construct a BN to compute the solution to a query, they rely on the fact that the solution can be determined by looking at a finite number of variables.

4.1 Iterative Approximation

Clearly, what is needed is a method for determining an approximate solution to a query by looking at a finite number of nodes. The discussion of Section 3 suggests that we can obtain an approximate solution by looking at a BN fragment that contains the query and evidence variables. We can develop an *anytime algorithm* that repeatedly constructs larger and larger BN fragments, and solves each one in turn to get better and better approximate solutions. In order to develop the algorithm, we make the following definitions:

Definition 4.1: Let \mathcal{K} be a probabilistic relational KB. A query Q is defined by a set \mathbf{Q} of *query variables* of \mathcal{K} , a set \mathbf{E} of *evidence variables* of \mathcal{K} , and an assignment \mathbf{e} to \mathbf{E} . Let \mathbf{X}_i be the set of variables of \mathcal{K} of length $\leq i$, together with \mathbf{Q} and \mathbf{E} . Let \mathbf{Y}_i be the roots of \mathbf{X}_i . We define π_i to be the set of distributions π over \mathbf{Q} such that $\pi = (\varphi \cdot B_{\mathbf{X}_i})(\mathbf{Q} \mid \mathbf{E} = \mathbf{e})$, for some distribution φ over \mathbf{Y}_i . We define π_{*i} to

be the function on $Val[\mathbf{Q}]$ defined by $\pi_{*i}(\mathbf{q}) = \inf\{\pi(\mathbf{q}) : \pi \in \pi_i\}$. Similarly, π_i^* is defined by $\pi_i^*(\mathbf{q}) = \sup\{\pi(\mathbf{q}) : \pi \in \pi_i\}$. We define $\hat{\pi}_i$ to be $(\varphi^0 \cdot B_{\mathbf{X}_i})(\mathbf{Q} \mid \mathbf{E} = \mathbf{e})$, where φ^0 is the uniform distribution over \mathbf{Y}_i . We call π_{*i} , π_i^* and $\hat{\pi}_i$ the *i -th order lower bound*, *upper bound* and *approximate solution to Q* respectively. ■

Informally, π_i represents the set of solutions to the query that are consistent with a BN fragment consisting of variables associated with entities at most i steps away from the top-level instance. π_{*i} and π_i^* are functions that assign lower and upper bounds to the probability of any assignment to the query variables. They are not themselves distributions. However, the set π_i is closed and convex, and any distribution lying between the bounds is attained by some element in π_i . In particular, if we are interested in the probability of a particular assignment \mathbf{q} to \mathbf{Q} , then we know that the probability must lie in $[\pi_{*i}(\mathbf{q}), \pi_i^*(\mathbf{q})]$, and that every value in between the bounds is consistent with the BN fragment.

We can construct an anytime algorithm by iteratively computing bounds of ever increasing order. The algorithm can be terminated at any time, and it returns the highest order bounds computed so far. The following theorem tells us that these bounds are valid, and that bounds from later iterations are at least as good as bounds from earlier iterations. It also tells us that the bounds eventually converge to the best possible bounds on the solution.

Theorem 4.2: Let \mathcal{K} be a probabilistic relational KB, and Q a query. Let b be the length of the longest query or evidence variable of Q . If $j \geq i \geq b$, then for any model μ of \mathcal{K} ,

$$\pi_{*i} \leq \pi_{*j} \leq \mu(\mathbf{Q} \mid \mathbf{E} = \mathbf{e}) \leq \pi_j^* \leq \pi_i^*.$$

Furthermore, if π is a distribution over \mathbf{Q} satisfying $\lim \pi_{*i} \leq \pi \leq \lim \pi_i^*$, then there exists a model μ for \mathcal{K} such that $\mu(\mathbf{Q} \mid \mathbf{E} = \mathbf{e}) = \pi$. In particular, if Q has a unique solution, the bounds will converge to the solution.

The condition that $i \geq b$ ensures that the set \mathbf{X}_i is self-contained. The first statement is a straightforward result of Lemma 3.4, while the proof of the second statement is similar to that of Theorem 3.7.

There are several ways to compute the bounds π_{*i} and π_i^* . One is an exact computation, based on the linear programming approach of (Nilsson 1986). This approach is expensive, as it ignores all structure in the BN fragment. Another is the approximation algorithm of (Draper & Hanks 1994), which performs standard BN inference for interval-valued probabilities. Both approaches are substantially more expensive than standard BN inference. A cheaper alternative is to compute the approximate solution $\hat{\pi}_i$, which is a standard BN inference computation on $B_{\mathbf{X}_i}$. The approximate solution always lies within the bounds, and if Q has a unique solution, then the approximate solution will converge to the true solution as i tends to infinity.

4.2 Exploiting Structure

There are two problems with the algorithm described above. First, the amount of work done in the i -th iteration of the

approximation algorithm may grow exponentially in i . In the i -th iteration, the algorithm considers variables associated with objects at most i steps away from the top-level instance. In the genetic example, the number of such objects, and therefore the number of variables considered, is $\Theta(2^i)$. Second, the algorithm fails to exploit the object structure of the domain: the encapsulation of information within individual objects, so that only small amounts of information need to be passed between objects, and reuse of computation between different instances of the same class. It was shown in (Pfeffer *et al.* 1999) that exploiting this structure can lead to major speedup of inference. In this section we extend these techniques to an anytime approximation algorithm. Somewhat surprisingly, exploiting the object structure also serves to address the exponential blowup.

The basic idea of a structure-exploiting inference algorithm for relational probability models is to treat each object as a query-answering entity. An object answers a query by recursively issuing queries to related objects, receiving answers to those queries, and combining the answers together with the local probability models of its simple attributes to produce an answer to the original query.

Our algorithm, called **Iterative Structured Variable Elimination (ISVE)**, takes five arguments: a class C on which the query is asked; a set of attribute chains σ representing the query variables; a set of attribute chains ρ representing the evidence variables; the evidence $e \in \text{Val}[\rho]$; and the desired solution quality i . **ISVE** returns a probability distribution over the values of the query chains σ .

For each query Q on class C , the cache maintains the highest order solution computed so far, together with its order j . If $i \leq j$, the cached solution is returned. Otherwise, if $i = 0$, a trivial approximation is returned. If neither of these conditions hold, the algorithm proceeds as follows. For each complex attribute A , it asks a recursive query of the object related by A , with a requested solution quality of $i - 1$. The query chains passed to these recursive queries come from two sources. One is the original set of query chains σ . If $A.\sigma'$ is a chain in σ , σ' is a query chain for the recursive call on A . (Evidence chains are passed similarly.) The second is the local probability models of the simple attributes of C . If some simple attribute B has a chain $A.\sigma'$ as a parent, σ' is added as a query chain for the recursive call. The result of the recursive query is a distribution over the chains σ' .

After solving the recursive queries, **ISVE** computes a factor for each of the simple variables of C , conditioning on the evidence where necessary. Finally, the results of the recursive queries and the factors for the simple variables are combined, and standard BN variable elimination is used to eliminate all but the query chains σ from this set. The result is a probability distribution over σ , which is returned as the solution to the query.

The **ISVE** algorithm can be used both for computing i -th order bounds $[\pi_{*i}, \pi_i^*]$ or the i -th order approximate solution $\hat{\pi}_i$. When computing $\hat{\pi}_i$, the zero-order approximation is just a uniform distribution over σ , and the **VE** procedure at the end is standard BN variable elimination. When computing bounds, the zero-order approximation is the trivial $[0, 1]$ bound.

The **ISVE** algorithm gains the standard advantages of structure-based inference — it is able to exploit encapsulation and reuse of inference between different instances of the same class. As it turns out, there is another advantage in the context of an iterative anytime approximation algorithm — the reuse of computation between different iterations.

Example 4.3: Consider the simple query where we wish to compute a probability distribution over $Fred.Phenotype$. We begin by requesting a first-order approximation to this distribution, with a call to **ISVE**(Person, {Phenotype}, \emptyset , \emptyset , 1). The third and fourth arguments are empty because there is no evidence. No solution to this query is found in the cache, so the algorithm proceeds by making recursive queries on the complex attributes of Person. The imports of the Mother attribute are M-Chromosome and P-Chromosome, so a recursive call is made to **ISVE**(Person, {M-Chromosome, P-Chromosome}, \emptyset , \emptyset , 0). No solution is found in the cache, but this time the requested order is 0, so a zero-order approximation is immediately returned, and stored in the cache. The original computation continues with a recursive call on the Father attribute. This call is exactly the same as the one on the Mother attribute, and the cached solution is returned, illustrating the reuse of computation between different objects in the model. Since all the recursive calls have been returned, the **VE** computation can be performed in the top-level query, and a first-order approximation is returned for the original query.

The anytime algorithm then continues to ask for a second-order approximate solution to the top-level query, beginning with a call to **ISVE**(Person, {Phenotype}, \emptyset , \emptyset , 2). This results in a recursive call on the Mother attribute to **ISVE**(Person, {M-Chromosome, P-Chromosome}, \emptyset , \emptyset , 1), which in turn results in a recursive call to **ISVE**(Person, {M-Chromosome, P-Chromosome}, \emptyset , \emptyset , 0), and the zero-order solution to this query is in the cache from the previous iteration, so it returns immediately. In general, only two new calls to **UncachedISVE** are made on each iteration, one with the query chain Phenotype, and one with the chains M-Chromosome and P-Chromosome. ■

In the genetic example, the amount of work done in each iteration is constant. This is in contrast to the unstructured algorithm in which the amount of work done in each iteration grows exponentially. Is it always the case that after a certain number of iterations, the amount of work performed by **ISVE** becomes constant? The answer is yes, for our language. The main insight is that, for a given KB in our language and a given top-level query, there are only finitely many different queries that can be generated. In fact, for our simple language the number of distinct queries generated is at most $a + qb$ where a is the total number of complex attributes, q is the number of query and evidence variables of the top-level query, and b is the length of the longest query or evidence variable. It follows that there must be some depth L such that all the queries at depth L also appear at a shallower depth. Therefore, no query at depth L will ever be expanded, no matter how high the requested order for the original query. The amount of work performed from the L -th iteration onwards is constant.

Theorem 4.4: *If \mathcal{K} is a probabilistic relational KB, and Q is a query on \mathcal{K} , the asymptotic complexity of computing an n -th order approximation to the solution of Q using the Iterative SVE algorithm is linear in n .*

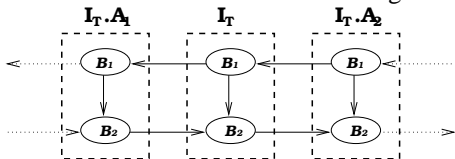
5 More Expressive Languages

So far we have considered a fairly simple language for relational probability models, that is sufficient to bring out the key points arising from infinite recursion. There are many natural extensions to the expressive power of the language, and here we consider a number of them.

While space considerations prevent us from giving details, it turns out that many of the obvious extensions do not change our results at all. These “free” extensions include: a finite set of named instances, with the knowledge-base specifying relations between them; multi-valued complex attributes, with or without uncertainty over the number of values for such an attribute; a class hierarchy, with uncertainty about types of instances; statements of the form A same-as σ , saying that for any instance I of C , the value of $I.A$ is the same as that of $I.\sigma$ (an appropriate stratification assumption needs to be made to obtain well-defined semantics); and *reference uncertainty*, where we allow uncertainty as to which chain the attribute A actually refers.

One language extension that does impact our results is to allow inverse statements of the form A inverse-of B . Such a statement means that $I.A = J$ implies that $J.B = I$. Inverse statements are an issue because they allow us to create KBs in which there are arbitrarily long chains of dependencies between two variables X and Y .

Example 5.1: Consider a KB consisting of a single class, with two complex attributes A_1 and A_2 , and two simple attributes B_1 and B_2 . The KB contains the statements A_1 inverse-of A_2 and A_2 inverse-of A_1 . Attribute B_1 has the parent $A_2.B_1$, while B_2 has the parents B_1 and $A_1.B_2$. The infinite BN for this KB has the following structure:



Note that the inverse relationship allows information to be passed both ways along the chain. There are arbitrarily long dependency chains between $I_T.B_1$ and $I_T.B_2$, so that there is no finite self-contained set containing both. ■

We call a KB *normal* if every finite set of variables is a subset of a finite self-contained set. It is possible to weaken our semantics so that it still imposes constraints on non-normal KBs; some of our results (e.g., Theorem 4.4) still carry through to this case. We omit details for lack of space.

Another very useful language extension is to allow an object to *pass arguments* to a related object, by *binding* the values of attributes of the related object. In all the languages considered so far, an object pulls values from related objects by mentioning values of other objects as parents of its simple attributes. Allowing an object to push values to related objects is a significant extension to the expressive power of

the language. It allows us to describe *stochastic functions* in the style of (Koller, McAllester, & Pfeffer 1997) (denoted KMP from now on).

Example 5.2: A *stochastic context-free grammar (SCFG)* is a context free grammar in which a probability distribution is provided over the productions associated with each non-terminal. An SCFG defines a probability distribution over the generation of strings in an obvious way.

An SCFG can be specified in our language as follows. There is a **String** class, with three attributes, the simple attribute **First**, ranging over the terminal symbols, the boolean simple attribute **Is-Empty**, and the complex attribute **Rest** of range type **String**. **String** is an abstract class — we will define some subclasses and specify their probability models. There is a **Concat** subclass of **String**, representing the concatenation of two strings. This subclass illustrates how something that we normally think of as a function can be represented in our language. Functionally, we can define **Concat** using the following recursive equations: $\text{Concat}([], ys) = ys$ and $\text{Concat}([x:xs], ys) = [x, \text{Concat}(xs, ys)]$. We define **Concat** in our language as follows. It has two complex attributes **Left** and **Right** of type **String**, corresponding to the arguments of the function. The **Is-Empty** attribute of **Concat** depends on **Left.Is-Empty** and **Right.Is-Empty**, and its CPF is the **and** function. The **First** attribute of **Concat** depends on **Left.First**, **Left.Is-Empty** and **Right.First**, and its CPF specifies it to be equal to **Left.First** if **Left.Is-Empty** is false, otherwise it is equal to **Right.First**. **Concat** has a **Subcall** attribute, representing the recursive call shown in the second part of its definition. In order to specify the recursive call properly, we need to be able to pass its arguments. This is where the binding mechanism comes in. We bind the value of **Subcall.Left** with **Left.Rest**, while **Subcall.Right** is bound to **Right**. To pass the results of the recursive call back upwards, we use the same-as mechanism described earlier. The value of the **Rest** attribute of **Concat** is the same as **Subcall**, if **Left.Is-Empty** is false, otherwise it is the same as **Right.Rest**.

Using the **String** and **Concat** classes, we can now construct classes for every symbol in an SCFG. The class corresponding to a terminal symbol simply defines a string of length 1 containing the given symbol, while the class for a non-terminal contains a string for each of the possible productions for that non-terminal, and chooses the appropriate string with the given probability. The string corresponding to a production is formed by applying **Concat** to the strings associated with each of the symbols in the production. We omit the details. ■

Basically, any probability model that can be described as a stochastic computational process can be defined in our language, once we allow argument passing via the binding mechanism. This language is much more expressive than those considered previously, but our semantics applies equally well to this language as to the others. Even when the stochastic computation is not guaranteed to terminate, the semantics still defines constraints on the probability distributions over variables at the roots of the process. This is

much stronger than the semantics defined in KMP, which required that the stochastic computation be guaranteed to terminate in order for the probability model to be well-defined.

In KMP, it was shown how Bayesian inference can be performed on stochastic programs by exploiting the structure of the program. In particular, it was shown that the inference algorithm mimics the dynamic programming behavior of the *inside algorithm* for SCFGs. The **ISVE** algorithm achieves the same behavior on SCFGs as that of KMP. The dynamic programming effect is achieved by caching. **ISVE** also has several advantages over the algorithm of KMP. For one, it is an anytime approximation algorithm, whereas the KMP algorithm does not produce any answer when the stochastic program is not guaranteed to terminate in a finite amount of time. For example, the KMP algorithm can answer a query asking whether an SCFG produces the string s , but it cannot answer a query asking whether the SCFG produces a string *beginning with* s , whereas ours can. In addition, our algorithm can be shown to achieve much better performance on programs that define Bayesian networks.

As was the case with inverse statements, allowing argument passing into the language makes it possible to pass information back and forth between two entities, allowing the creation of non-normal KBs such as that of Example 5.1. For normal KBs with argument passing, Theorems 3.7 and 4.2 continue to hold. However, Theorem 4.4 no longer holds, even for normal KBs. The reason is that this theorem relies on the fact that only a finite number of distinct queries can be generated from a given top-level query. Once we allow argument passing into the language, it is possible to generate queries of arbitrary complexity. This is hardly surprising given the expressive power of the language — it is Turing complete, in that it is capable of defining any stochastic computation. Despite the fact that Theorem 4.4 does not hold, the benefits of reusing computation between iterations are still obtained by the **ISVE** algorithm. In fact, the conclusion of the theorem still holds for many KBs in this more expressive language. In particular, it holds for queries on SCFGs. For example, consider a query over whether a grammar generates a string beginning with s . Each iteration of the algorithm considers longer and longer derivations that might produce a string beginning with s . The types of queries generated are whether a certain non-terminal generates a substring of s , or a string beginning with a substring of s . There are only finitely many such queries, and after a certain number of iterations all the different queries will have been expanded, and the amount of work performed on each iteration becomes constant.

6 Conclusions and Future Work

Recursive probability models are common and natural. In this paper, we have presented an elegant framework for dealing with such models. We provided a natural measure-theoretic semantics that extends the semantics of Bayesian networks, and showed that every normal KB has a model under this semantics. We also presented the **ISVE** algorithm for anytime approximate inference. The algorithm exploits the relational structure of a domain, thereby gaining great

benefits in the context of anytime inference.

There are a number of important questions about recursive probability models that remain to be explored. Some key questions are: Is there a general theory that can tell us when a model has a unique distribution? Can conditions be found under which Theorem 4.4 holds even for models with argument-passing? What kinds of conclusions can one draw from KBs that are not normal, and does Theorem 3.7 hold for such KBs?

We believe that applying finite or iterative computation to an infinite model is a fundamental AI technique. An example is game tree search, in which a finite portion of a possibly infinite game tree is expanded in order to determine the best move. Hughes (Hughes 1989) has argued that allowing the tree to be specified in its natural infinite form, and then applying a finite lazy computation to it, is the most elegant and modular way of expressing this process. The same holds true for probabilistic reasoning. Rather than forcing our possible worlds to be finite by applying arbitrary restrictions to our representation language, we can freely allow them to be infinite, and use the techniques of lazy evaluation and anytime approximation to deal with them.

Acknowledgements

We would like to thank David McAllester for useful discussions. This work was supported by ONR contract N66001-97-C-8554 under DARPA's HPKB program.

References

- Draper, D., and Hanks, S. 1994. Localized partial evaluation of belief networks. In *Proc. UAI*.
- Enderton, H. 1972. *A Mathematical Introduction to Logic*. Academic Press.
- Hughes, J. 1989. Why functional programming matters. *The Computer Journal* 32(2):98 – 107.
- Koller, D., and Pfeffer, A. 1998. Probabilistic frame-based systems. In *Proc. AAAI*.
- Koller, D.; McAllester, D.; and Pfeffer, A. 1997. Effective Bayesian inference for stochastic programs. In *Proc. AAAI*.
- Ngo, L., and Haddawy, P. 1996. Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*.
- Nilsson, N. 1986. Probabilistic logic. *AIJ* 28(1):71–87.
- Pfeffer, A.; Koller, D.; Milch, B.; and Takusagawa, K. 1999. SPOOK: A system for probabilistic object-oriented knowledge representation. In *Proc. UAI*.
- Pfeffer, A. 2000. *Probabilistic Reasoning for Complex Systems*. Ph.D. Dissertation, Stanford University.
- Wellman, M.; Breese, J.; and Goldman, R. 1992. From knowledge bases to decision models. *The Knowledge Engineering Review* 7(1):35–53.