In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98),* pages 580-587, Madison, Wisconsin, July 1998

Probabilistic frame-based systems

Daphne Koller
Computer Science Department
Stanford University
Gates Building, 1A
Stanford, CA 94305-9010
koller@cs.stanford.edu

Avi Pfeffer
Computer Science Department
Stanford University
Gates Building, 1A
Stanford, CA 94305-9010
avi@cs.stanford.edu

Abstract

Two of the most important threads of work in knowledge representation today are frame-based representation systems (FRS's) and Bayesian networks (BNs). FRS's provide an excellent representation for the organizational structure of large complex domains, but their applicability is limited because of their inability to deal with uncertainty and noise. BNs provide an intuitive and coherent probabilistic representation of our uncertainty, but are very limited in their ability to handle complex structured domains. In this paper, we provide a language that cleanly integrates these approaches, preserving the advantages of both. Our approach allows us to provide natural and compact definitions of probability models for a class, in a way that is local to the class frame. These models can be instantiated for any set of interconnected instances, resulting in a coherent probability distribution over the instance properties. Our language also allows us to represent important types of uncertainty that cannot be accomodated within the framework of traditional BNs: uncertainty over the set of entities present in our model, and uncertainty about the relationships between these entities. We provide an inference algorithm for our language via a reduction to inference in standard Bayesian networks. We describe an implemented system that allows most of the main frame systems in existence today to annotate their knowledge bases with probabilistic information, and to use that information in answering probabilistic queries.

1 Introduction

Frame representation systems (FRS's) are currently the primary technology used for large scale knowledge representation in AI [8, 3, 7]. Their modular organization according to cognitively meaningful entities and their ability to capture patterns common to many individuals provide a convenient language for representing complex structured domain models. One of the most significant gaps in the expressive power of this type of framework is its inability to represent and reason with uncertain and noisy information. Uncertainty is unavoidable in the real world: our information is often inaccurate and always incomplete, and only a few of the "rules" that we use for reasoning are true in all possible cases.

In the "propositional" setting, this problem has largely been resolved over the past decade by the development of probabilistic reasoning systems, and particularly Bayesian networks [10]. A Bayesian network (BN) is a representation of a full joint distribution over a set of random variables; it can be used to answer queries about any of its variables given any evidence. A BN allows a complex distribution to be represented compactly by using the locality of influence in our model of the world. But, like all propositional systems, the applicability of BNs is largely limited to situations that can be encoded, in advance, using a fixed set of attributes. Thus, they are inadequate for large-scale complex KR tasks.

Building on our recent work [6, 5], we propose a representation language that integrates frame-representation systems and Bayesian networks, thereby providing the first bridge between these two very different threads of work in KR. The key component in our representation is the annotation of a frame with a probability model. This probability model is, broadly speaking, a BN representing a distribution over the possible values of the slots in the frame. That is, each simple slot in the frame is annotated with a local probability model, representing the dependence of its value on the values of related slots. For example, in a frame representing a PhD student, the value of the slot *years-to-graduation* may depend on the slot *year* and the slot chain *advisor.picky*.

As we can see even from this simple example, by building on standard FRS functionality, our approach provides significantly more expressive power than traditional BNs. For example, by allowing the probability model of a slot to depend on a slot chain, we allow the properties of one instance in the model to depend on properties of other related instances. We can also use the standard class hierarchy of the FRS to allow the probability model of a class to be used by multiple instances of that class, and to allow inheritance of probability models from classes to subclasses, using the same mechanism in which slot values are currently inherited. Finally, by making domain individuals first-class citizens in our framework we can also express a new and important type of uncertainty called *structural uncertainty*. We can have a probabilistic model expressing our uncertainty about the set of entities in our model, e.g., the number of PhD students in a department. We can also represent uncertainty about relations between entities, e.g., which of several conferences a paper appeared in.

We provide a probabilistic inference algorithm for our language based on an approach known as *knowledge-based* *model construction*. The algorithm takes a knowledge base in our language, including a set of instances, and generates a standard BN which can then be queried effectively for our beliefs about the value of any slots.

Our probability model is expressed using standard frame representation techniques such as facets and value restrictions. This property is important, since it allows our approach to be used with virtually any frame system, and thereby to annotate existing KBs with probabilistic information. In particular, we have implemented a system based on our approach, capable of interacting with most existing FRS's via *OKBC* [2], an emerging standard for FRS interoperability.

Our work is a signficant improvement over previous approaches to combining first-order logic and Bayesian networks. Most of the attempts in this direction (e.g., [12, 11, 9]) use probabilistic Horn clauses as the basic representation. The choice of Horn clauses as an underlying language already dictates some of the properties of the representation, e.g., its inability to encapsulate an object and its properties within a cognitively meaningful frame. Moreover, the use of structural uncertainty in this framework typically causes combinatorial blowup of the resulting models, leading most approaches to outlaw it entirely. Our framework also overcomes some major limitations of our earlier proposals [6, 5], by allowing both structural uncertainty (absent in the first) and probabilistic dependencies between instances (absent in the second). It also provides the crucial ability, absent in both, to create complex models containing many instances that are connected to each other in a variety of ways.

2 Basic representation

We begin with some basic terminology for frame systems. The terminology varies widely from system to system. In this paper we adopt the language and basic knowledge model of the OKBC protocol [2].

The basic unit of discourse in a frame system is a *frame*. A frame has a set of *slots*, each of which may have *slot values* or *fillers*. Formally, a slot represents a binary relation on frames; if the filler of slot A in frame X is frame Y, then the relation $A(X,Y)$ holds. In general slots may be single-valued or multi-valued. In this section we assume that slots are single-valued. This assumption will be relaxed in Section 4. A *slot-chain* is a sequence of zero or more slots separated by periods. A slot-chain represents a binary relation: the slot-chain $A.\sigma$ where A is a slot and σ is a slot-chain denotes the relation $\{(X,Z) \mid A(X,Y) \wedge \sigma(Y,Z)\}$. A slot in a frame may have associated *facets*. A facet is a ternary relation: if the facet value of facet F on slot A in frame X is Y, then the relation $F(X,A,Y)$ holds. A standard facet is value-type, which specifies a value restriction on the values of a slot. The value-type of a slot will be called its *type*.

The two main types of frames are *class frames*, representing sets of entities, and *instance frames*. The class frames are organized in an is-a hierarchy, where one class may be a *subclass* of another (its *superclass*). The slots of a class frame may be *own slots*, which describe a property of the class itself, and *template slots*, which are slots inherited by all instances and subclasses of the class. The facets associated with template slots are *template facets*, and are also inherited. An instance or subclass may override the values of inherited slots or facets.

Probabilistic information is incorporated into a frame KB by annotating class frames with local probabilistic models. A class frame that has been so annotated is called a *p-class*. A p-class has a set of template slots, each with a value-type facet. Depending on the type, a slot is either *simple* or *complex*. The type of a complex slot is another p-class. The type of a simple slot is an explicitly enumerated list of possible values for the slot. For example, the phd-student p-class may have a simple slot *year*, whose type is $\{$*1st, 2nd, 3rd, 4th–6th, tenured*$\}$, and a complex slot *advisor* whose type is the p-class professor. A p-class may also have other slots that do not participate in the probability model, whose type is neither of the above. For example, phd-student may also have the slot *name*, which does not have an associated probability model. This feature allows existing KBs to be annotated with probabilistic information, without requiring a complete redesign of the ontology.

A simple slot is very much like a node in a Bayes net. It has a range of values, a set of parents, and a CPT. A p-class specifies a probability model for its simple slots using two special-purpose facets: parents and distribution. Facets are a natural place to put a probability model, since such a model can be viewed as a generalization of a value restriction: not only does it specify a range of possible values, but also a distribution over that range. The parents facet lists the slots on which the value of this slot depends. Each parent is specified by a slot-chain referring to some other simple slot. More precisely, let X be a p-class and A a simple slot. The parents facet of A is a list of slot chains $[\sigma_1,\ldots,\sigma_n]$, such that $X.\sigma_i$ refers to a simple slot. For example, in the phd-student p-class, *year* may have the parent [*age*], while the parents of *years-to-graduation* may be [*year*, *advisor.picky*]. The distribution facet specifies the conditional probability distribution over values of the slot given values of its parents. The conditional distribution is specified using a conditional probability table (CPT) as in Bayesian networks. For each combination of values of its parents, the CPT provides a probability distribution over values of the slot. For the purposes of this paper, we assume that the CPTs are represented as fully specified functions of parent values. More compact representations such as noisy-or can easily be accomodated within our framework.

The probability model of a complex slot is simply described by its p-class Y. However, each complex slot A also has an additional facet called imports, whose value is a list of slots in Y. This list, called the *import list* of A, is the list of slots of Y that are visible within X. We require that if $A.B.\sigma$ (for a possibly empty slot chain σ) is a slot chain appearing within X, then B must be in the import list of A.

Once a probability model has been specified for a p-class, the p-class can be used just like any other class frame. One can create instances of the class, which will inherit all of its template slots and facets. In particular, the probability distribution over values of slots of the instance will be as described in the p-class. Similarly, the inheritance mechanism of a frame system can be used to make one p-class a subclass of another. A subclass can extend the definition of the superclass as well as overwrite parts of it. In particular, a subclass can redefine the probability model of one or more of the slots. For example, we can define associate-professor to be a subclass of professor, and overwrite the distribution over *salary* to one that is appropriate to the more specific class. Another important aspect of subtyping is that an instance of a subclass is also an instance of the superclass, so that it can fill a slot whose type is the superclass. For example, in a particular instance of phd-student, the value of the *advisor* slot may be specified to be an instance whose class is associate-professor.

Values can be assigned to an own slot of an instance frame either directly or by assignment to a template slot at the class level. Both types of assignments are interpreted in the same way. An assignment to a simple slot is interpreted as observing the value of the slot, thereby conditioning the probability distribution for the instance. This conditioning process may result in a change in our beliefs for other related slots. Consider, for example, a subclass graduating-phd-student of phd-student which assigns 1 to the slot *years-to-graduation*. Then the conditioning process will result in a new probability model for any instance I of this subclass; in particular, our beliefs about I.*year* and I.*advisor.picky* will both change, as will our beliefs about other related slots.

An assignment to a complex slot specifies that the value of that slot is another particular instance. Thus, complex networks of inter-related frames can be created, such as students who share an advisor, and students of different advisors in the same department. Such an assignment at the class level results in all of the class instances having the same frame as their value for that slot.

One of the features of a probabilistic frame system is that related frames can influence each other. We have already seen one mechanism for such interactions: since a parent of a slot is a slot-chain, the value of a simple slot may be influenced probabilistically by the value of a slot in another frame. This mechanism, however, only allows a frame to be influenced by related frames, but not to influence them in turn. We resolve this difficulty by utilizing a basic feature of most FRS's—*inverse slots*.

Let X and Y be two class frames, A a slot of X with type Y, and B a slot of Y with type X. Then A and B are *inverse slots* if, for every instance I of X, if $I.A = J$ then $J.B = I$, and vice versa. Thus, we view an assignment of a specific instance frame J to a slot $I.A$ as encompassing the corresponding assignment of I to $J.B$. For that reason, we do not allow assignments of values to slots such as A at the class level; otherwise, for any given frame J of class Y, the value of $J.B$ would be the set consisting of every frame of class X, a model which is too unwieldy to deal with.

Inverse slots allow either of the frames to refer to slots in the other, thereby allowing probabilistic dependencies in both directions. Allowing such intertwined dependencies without restriction could lead to horribly complex interactions between two frames. In particular, it could lead to a cyclic chain of influences that has no coherent probability model. Therefore, one of the two inverse slots—say $X.A$—is designated to be the *primary* direction while the other—$Y.B$—is *secondary*. Similarly, X is called the primary frame, while Y is the secondary frame. A primary inverse slot such as A in X has a parents facet just like a simple slot, i.e., it is a list of slot-chains in X. Intuitively, the parents of A are the only slots of X that are *exported* to Y via B. More precisely, the parent list of A in X must be identical to the import list of B in Y. Thus, the flow of influence between the two frames is neatly regulated: The parents of A in X can influence any of the slots in Y; some of those can, in turn, influence other slots in X that are "downstream" from A.

For example, suppose we decide that the *thesis* slot of phd-student should be an inverse slot, with its inverse being the *author* slot of phd-thesis. The slot *thesis* is designated to be primary, and is given the parent *field* in phd-student. Then *field* is visible within the phd-thesis class, so that for example, *jargon-content* may depend on *author.field*. Other slots of phd-student may depend on slots of thesis; thus, for example, *job-prospects* may depend on *thesis.quality*, which would therefore have to be on the import list of *thesis*.

Inverse slots serve a dual purpose in our language. As we said, they allow bidirectional dependencies between two instances. But they also allow our probabilistic models to be *multicentered*. If, as above, $X.A$ and $Y.B$ are inverses, and we define an instance from class X, it immediately implies the existence of a corresponding instance from class Y. Alternatively, we could start modeling with an object of class Y, and guarantee that the corresponding X will exist. Thus, we can define a model centered around whatever entities are of interest to us in our context.

3 Semantics

In this section we present a semantics for probabilistic frame knowledge bases. For a given KB with a set of class and instance frames, our semantics defines a probability distribution over the slot values of the instance frames (and of some other related instance frames). In order to define a coherent probability distribution, our frame KB must satisfy several conditions. The basic theme of the conditions is familiar from the realm of Bayesian networks: our dependency model must be acyclic. However, since there may be complicated chains of dependencies both within a frame and between frames, and on both the class and instance levels, we need to develop some tools to reason about dependencies.

Definition 3.1: A *dependency* is a pair $X.A \leftarrow Y.B$, where X and Y are frames (not necessarily distinct), A is a slot of X and B is a slot of Y. We say that $X.A \leftarrow Y.B$ holds if

- A is a simple slot of X, $Y = X$ and $B.\sigma$ is a parent of A;
- A is a complex slot of X, B is in the import list of A, and Y is either an instance frame assigned to $X.A$ or the p-class frame which is the value type of $X.A$. ∎

Intuitively, a dependency $X.A \leftarrow Y.B$ asserts that for every instance frame I consistent with X there exists an instance frame J consistent with Y such that $I.A$ depends on $J.B$. (If X is itself an instance frame I, then only I is consistent with X; if X is a class, then any instance of that class is consistent with X.) Note however, that our definition of dependencies only considers the first slot in a chain on which a slot depends; thus, it makes only a first-level partition of dependency. It is a conservative overestimate of the true dependency model, since if $X.A \leftarrow Y.B$, it is not necessarily the case that $X.A$ depends on every slot-chain

$Y.B.\sigma$. While it is fairly straightforward to refine our notion of dependency, we have found our definition to be adequate for most purposes.

Definition 3.2: A *dependency chain* is a list $X_1.A_1 \leftarrow X_2.A_2 \leftarrow \cdots$ such that, for each i, $X_i.A_i \leftarrow X_{i+1}.A_{i+1}$. A *dependency cycle* is a dependency chain that begins and ends with the same slot. ∎

Dependency cycles reflect potential problems with our model. (Although, as indicated by our discussion above, some correct models may appear to be problematic simply because of our overestimate for probabilistic dependencies.) A dependency cycle containing $I.A$, where I is an instance frame, corresponds to a possible chain of dependencies through which $I.A$ depends on itself. Such a cyclic dependency, if it exists, prevents us from defining a coherent probability model. A dependency cycle containing $X.A$ for some class X means that for every instance I_1 of X there is some instance I_2 of X such that $I_1.A$ depends on $I_2.A$. In some cases, I_1 and I_2 are necessarily the same instance; such cases are called *truly cyclic*. In others, however, they are distinct instances of the class X. These cases can also be problematic, as they may represent an infinite dependency chain beginning with $I_1.A$: $I_1.A$ depends on $I_2.A$ which depends on some $I_3.A$, etc. Such models also do not typically have well-defined probabilistic semantics.

We conclude from this discussion that we want to disallow all dependency cycles.[1] Some types of dependency cycles are easy to prevent using purely local considerations. Specifically, we can build, for each class X, a *dependency graph* for X. This graph contains all the slots of X, with an edge from B to A if the dependency $X.A \leftarrow X.B$ holds. Clearly, if we want to avoid dependency cycles, this graph should be acyclic. Indeed, our care in designing the dependency model for inverse slots implies that if we make all class dependency graphs acyclic, we avoid any truly cyclic dependency chains at the class level. Formally, if we define a *class-level dependency chain* to be one in which all the X_i's are p-classes, we obtain the following theorem:

Theorem 3.3: *If we have a knowledge base in which all class dependency graphs are acyclic, then there are no truly cyclic class-level dependency chains.*

However, as we discussed, even dependency chains that are not truly cyclic can result in incoherent models. In addition, we have not eliminated instance-level dependency chains that are truly cyclic. Unfortunately, the general problem is not so easy to prevent using purely local constraints. However, we can detect whether or not the KB contains a dependency cycle by building a more global directed graph $\mathcal{G}$, called the *dependency graph* of the KB. The nodes of $\mathcal{G}$ are all $X.A$ where X is a p-class or named individual frame and A is a slot of X. There is an edge from $Y.B$ to $X.A$ if the dependency $X.A \leftarrow Y.B$ holds. Clearly, the KB contains a dependency cycle iff $\mathcal{G}$ is cyclic.

For a KB that contains no dependency cycles, our goal now is to define a probability distribution over instantiations to frames, i.e., assignments of values to the slots of the frames. Several issues combine to make such a definition difficult.

The most obvious idea is to follow the approach taken in the semantics of Bayesian networks: we determine a set of random variables, and define a distribution over their joint value space. Unfortunately, our framework is too rich to make this approach appropriate. As we mentioned, the set of instance frames that we can potentially refer to may be infinite. While one might be able to circumvent this particular problem, a more serious one manifests when we enrich our language with structural uncertainty in Sections 4 and 5. Then, the set of instance frames can also vary probabilistically, in a way that both depends on and influences the values of other random variables in the model.

We therefore define our semantics via a data generating process, that randomly samples values for the various frames in the model. The random sampling process implicitly defines a distribution over the different possible value assignments: the probability of a value assignment is the probability with which it is generated by the process. Note that, although a random sampling process can also be used as a stochastic algorithm for approximate inference, we are not proposing this approach; our sampling process is purely a thought experiment for defining the distribution. In Section 6, we show how a more standard process of exact inference can be used to effectively answer queries relative to this distribution.

The sampling process builds value assignments to slots of frames incrementally, as the different components are required. By allowing such partial assignments, we bypass the problem of going off on infinite sampling chains. The assumption of finite dependency chains guarantees that the sampling chains required to sample the value of any simple slot will always terminate.

Definition 3.4: A partial value ϑ for an instance frame is an assignment of values (of the appropriate type) to some subset of its simple slots, an assignment of instance frames (from the appropriate p-class) to some subset of its complex slots, and a partial value for each of these assigned instances. ∎

One final subtlety arises in the sampling construction. Some instance frames may have specific values pre-assigned to some of the their slots. Such an assignment can be done via an explicit statement for a named instance frame, or via a process of inheritance from a template slot of a class to which the instance belongs. As we explained in Section 2, the semantics of such assignments is to condition the distribution. To obtain the right semantics, we make the obvious modification to our data generating process. If, during the sampling process, a value is generated for a slot which is inconsistent with the observed value, we simply discard the entire partial value generated up to that point. It is easy to see [4] that the relative probability with which a partial value is generated in this data generating process is exactly the same as its probability conditioned on the observed slot values.

As we discussed, our sampling procedure builds up a partial value ϑ piece by piece, as the pieces are needed. Our main procedure, shown in Figure 1, is **Sample**(I, A), which samples the value of a single simple slot A of a single instance frame I. In order to sample A from the correct distribution, it must backward chain and sample other slots on which the

[1] Note that we are not disallowing infinite reference chains (chains of related instances), unless they imply infinite dependency chains.

```
Sample(I, A)
  If A has a value in ϑ then return
  Foreach parent σ of A
    If σ is a slot B in I then
      Sample(I, B)
    Else /* σ is of the form σ'.C */
      Let J := ComplexValue(I, σ')
      Sample(J, C)
  Choose(I, A)

Choose(I, A)
  Choose a value v for A
    according to P(A | Pa(A))
  Extend ϑ with I.A = v
  If A has a pre-assigned value v' then
    If v' ≠ v then fail
```

```
ComplexValue(I, σ)
  If σ is empty then
    Return I
  /* σ is of the form B.σ' */
  If I.B is assigned a value J in ϑ then
    Let K := J
  Else if I.B is pre-assigned a value J then
    Extend ϑ with I.B = J
    Let K := J
  Else
    Let Y be value-type(B)
    Create a new instance K of p-class Y
    Extend ϑ with I.B = K
    If B has an inverse B' in K then
      Extend ϑ with K.B' = I
  Return ComplexValue(K, σ')
```

Figure 1: Data generating sampling model

value of A depends. **ComplexValue**(I, σ) determines the value of the complex slot-chain $I.\sigma$, if necessary creating new instance frames to represent the values of complex slots. When the procedure returns, the partial value ϑ (a global variable in the procedure) contains a value for $I.A$.

Lemma 3.5 : *If the dependency graph is acyclic, then* **Sample**(I, A)*, executed from a partial value ϑ defines a probability distribution over extensions of ϑ with a value assigned to $I.A$. Furthermore, the distribution does not depend on the order in which the parents of A are examined.*

Proof: The basic steps in the proof are as follows. To prove the first part of the theorem, it suffices to show that the sampling algorithm terminates. This proof proceeds using a simple inductive argument over the length of dependency chains. To prove that the distribution is independent of the order, we observe that a simple slot is always generated from the same conditional distribution, regardless of when it is sampled, and that the failure conditions are also applied universally. ∎

We can now define a **SampleKB** procedure that uses **Sample**(I, A) to sample values for the slots of all named instances in the KB. If any call to **Sample** fails, the entire sampling process needs to be restarted. Once a value has been assigned to all simple slots of named instances, we have accounted for all evidence in the model, and therefore further sampling of other slots cannot possibly fail. Therefore the distribution we have obtained over the slots we have sampled is the final one.

Theorem 3.6: *If the dependency graph is acyclic then* **SampleKB** *defines a probability distribution over partial values ϑ which have values assigned to all simple slots $I.A$ for all named instances I.*

4 Multivalued slots and number uncertainty

To this point, we have assumed that every slot is single-valued. However, slots that take on multiple values are a fundamental concept in frame systems. The ai-professor p-class may have a multi-valued *papers* slot of type ai-paper. To simplify our discussion, we require multi-valued slots to be complex, and all values of the slot must be of the same p-class, as specified in the slot's value-type.

To allow other slots in a frame X to depend on the properties of a multi-valued slot A, we must present a way for a slot to depend on a set of slots. As the elements in the set cannot be referred to individually, we must refer instead to the properties of the set.

Definition 4.1: A *quantifier slot* for a multi-valued slot A has the form $\forall(A.\sigma \;:\; e)$, $\exists(A.\sigma \;:\; e)$, $\leq n(A.\sigma \;:\; e)$ or $\geq n(A.\sigma \;:\; e)$, where σ is a slot chain of A and e is an element of the value type of $A.\sigma$. The value-type of a quantifier slot is the set $\{$*true*$,$ *false*$\}$. ∎

Given a set of values for A, the value of a quantifer slot on $A.B$ has precisely the meaning that one would expect; for example, if for at least 10 values I of the *papers* slot $I.$*impact* has the value *high*, then the value of $\geq 10($*papers.impact* $:$ *high*$)$ is *true*. Note that the CPT of a quantifier slot is well-defined for any number of values of its multi-valued slot. On the other hand, no other slot can depend directly on the multi-valued slot, thereby avoiding the problem of defining general CPTs with a variable number of parents. A quantifier slot, on the other hand, is treated in the same way as a simple slot, so it may be used as a parent of another slot. Thus, for example, the *will-get-tenure* slot of the assistant-professor class may depend on the above quantifier slot.

So far, we have not specified the number of values that a given multi-valued slot can take. In many cases, e.g., the number of papers, this number is not fixed. Therefore, we would like to be able to model situations in which different numbers of papers are possible, and to represent our beliefs in these various possibilities. In other words, we would like to allow *structural uncertainty*—uncertainty over the set of entities in the world and the relationships between them. Uncertainty over the number of values of a multi-valued slot is a type of structural uncertainty called *number uncertainty*.

We can extend our language to represent number uncertainty by associating with each multivalued slot A of X a new *number slot num*(A), which ranges over some set of natural numbers $\{0, 1, \ldots, n\}$ (we assume that the number of values of every slot is bounded). The slot *num*(A) is treated just like any other simple slot; it has parents and distribution facets that describe its probability model. Thus, the number of values of A in X can depend on values of other slots of X and of related frames, and it can also be the parent of another slot. For example, ai-professor will have a *num*(*papers*) slot, whose value ranges from 0 to 50; *num*(*papers*) may depend on *productivity* and in turn influence *tired*.

As with the case of single-valued slots, a specific value I may be asserted for a multi-valued slot A of both a p-class and a named individual. We interpret such assignments as asserting that one of A's values is I. It does not prevent A from having other values; in fact, multiple values may be asserted for the slot. Such assignments do not eliminate our number uncertainty for this slot, but any case where the slot has fewer than the number of asserted fillers is eliminated; thus, we must condition *num*(A) to be at least the asserted number. To assert that A has only the values mentioned, we would need to explicitly assert a value for *num*(A).

It is interesting to examine the interaction between multi-valued slots and inverses. Assume that the *advisees* slot has an inverse *advisor* within the phd-student frame. If we now have a student instance frame I, then we automatically assert at least one value—the value I—for the *advisees* slot in the instance frame $I.advisor$. Thus, even if we have no other information whatsoever about this instance frame, it will not be a generic member of the professor class. The very fact that the professor is someone's advisor modifies its distribution by conditioning it on the fact that $num(advisees) \geq 1$. Note that the inverse slot may also be multi-valued. Many-many relations give rise to potentially infinite reference chains. For example, a paper may have several authors, who have all written several papers, and so on. However, due to the restrictions on the flow of influence between primary and secondary inverses, an infinite reference chain of this sort cannot lead to an infinite dependency chain.

Number uncertainty can be incorporated into our semantics quite easily. We need to add number and quantifier slots into the dependency graph. Number slots are treated just like simple slots: there is an edge into $X.num(A)$ for each of the parents of $num(A)$ in X. If $X.Q$ is a quantifier slot over $X.A$, there is an edge from $X.A$ to $X.Q$. Finally, we must make the value of $X.A$ depend both on $X.num(A)$ and the properties it imports from each of its fillers. If $X.A$ imports B, then we have $X.A \leftarrow Y.B$, where Y is the type of A, and $X.A \leftarrow I.B$ for every asserted value I of $X.A$.

The sampling process can easily be modified to account for multi-valued slots. When a multi-valued slot A needs to be sampled for the first time, we sample first a value n for $num(A)$. Let m be the number of asserted values for A. If $n < m$, the sample fails. Otherwise, $n - m$ new instances of the type of A are created, and the set of values of A is set to be the m asserted fillers and the $n - m$ new instances. Theorem 3.6 continues to hold.

5 Reference uncertainty

As we said, structural uncertainty allows us to represent distributions over models with different structures. Number uncertainty allows us to vary the *set* of instances in our model. In this section, we describe *reference uncertainty*, which allows us to vary the relations between the instances. For example, we may want the *conference* slot of the AI-paper class to be AAAI with probability 0.3, and another generic AI conference with probability 0.7; note that AAAI is not the value of a simple slot, but an instance frame itself. We extend our language to accomodate reference uncertainty by allowing some complex slots to be *indirect*. Each indirect slot A is associated with a *reference slot* $ref(A)$, a simple slot whose value dictates the value of the indirect slot.

Definition 5.1: If A is an indirect slot of type Y in p-class X, then $ref(A)$ is a simple slot in X whose value type is an enumerated set $\mathcal{R}$, each of whose values ρ is either: a named individual of type Y, a slot-chain of X whose type is Y, or the class Y itself.

In any instance I of X, the value of A is defined in terms of the value of $ref(A)$: if the value of $ref(A)$ is a named individual J, $I.A = J$; if the value of $ref(A)$ is a slot-chain σ, then $I.A = I.\sigma$; if $ref(A)$ is the p-class Y, then the value of A is a new instance of Y. ∎

A reference slot is treated just like any other simple slot, so it has parents and a CPT, and can influence other simple slots. A value can be assigned to a reference slot from within its value-type, and the value of the indirect slot will be determined by it. An indirect slot is treated like any other single-valued complex slot; it has an import list, and other slots can depend on the slots it imports. For technical reasons, we do not allow direct assignments to indirect slots, nor do we allow them to have inverses.

As with number uncertainty, reference uncertainty can be incorporated quite easily into our semantics. We need to add reference and indirect slots to the dependency graph. A reference slot is treated like any other simple slot; thus, its only dependencies are on its parents. An indirect slot A clearly depends on $ref(A)$, so we have $A \leftarrow ref(A)$. Since A is a complex slot, it also depends on the slots that it imports. However, because of reference uncertainty, we do not know the frame from which it imports those slots. Let B be some slot on the import list of A. To be safe, we need to account for every possible value ρ of $ref(A)$. Thus, for each $\rho \in \mathcal{R}$, if ρ is a named individual I, we have $X.A \leftarrow I.B$; if ρ is a slot chain $C.\sigma$, we have $X.A \leftarrow X.C$; if ρ is the class Y, we have $X.A \leftarrow Y.B$, denoting the fact A may import B from some instance of Y.

The sampling process requires a small change to AssignComplex to deal with indirect slots. When AssignComplex is called with an indirect slot, we first sample the value of the corresponding reference slot in the usual manner, and then assign the value of the indirect slot in the manner determined by the value of the reference slot. With this change, Theorem 3.6 continues to hold.

6 Inference

In the preceding sections, we presented a representation language and semantics for probabilistic frame systems. To complete the story, we now present a simple inference algorithm for answering probabilistic queries in such a system. Our algorithm can handle any instance-based query, i.e., queries about the values of slots of instances. For simplicity, we restrict attention to simple slots of named instances, as other queries can easily be reduced to these. The algorithm, called **ConstructBN**, is based on *knowledge-based model construction* [12], the process of taking a KB and deriving a BN $\mathcal{B}$ representing the same probability model. Standard BN inference can then be used to answer queries.

Nodes in the Bayes net $\mathcal{B}$ have the form $I.\sigma.A$ where I is an instance frame (not necessarily named), σ is a possibly empty slot chain, and A is a simple slot. The algorithm works by backward chaining along dependency chains, constructing the appropriate nodes in the BN if they do not already exist. More specifically, the algorithm maintains an open list $\mathcal{L}$ of nodes to be processed. Initially, $\mathcal{L}$ contains only the simple slots of named instances. In each iteration, the algorithm removes a node from $\mathcal{L}$, and processes it. When a node is removed from $\mathcal{L}$, it is processed in one of three ways: as a simple slot, as a slot chain, or as a quantifier slot.

Simple slots $I.A$ are processed as follows. For each parent $I.\sigma$, an edge is added from $I.\sigma$ to $I.A$ by a call to AddParent($I.A$, $I.\sigma$); if $I.\sigma$ is not already in $\mathcal{B}$, this routine adds $I.\sigma$ to $\mathcal{B}$ and $\mathcal{L}$. (Note that the parent list of any simple non-quantifier slot is fixed.) When all parents have been added, the CPT is constructed from the distribution facet of A.

A slot chain $I.B.\tau$ is processed as follows:

ProcessComplex($I.B.\tau$)
If B is indirect then
ProcessIndirect($I.B.\tau$)
Else
If B is assigned a value J in I then $K = J$
Else $K = X[I.B]$, where X is the type of B in I
AddParent($I.B.\tau$, $K.\tau$)
Set CPT of $I.B.\tau$ to copy the value of $K.\tau$

Essentially, if B is assigned a named individual J in I, then $I.B = J$. Otherwise, $I.B = X[I.B]$, an instance of X that does not appear anywhere else in the KB; roughly, $X[I.B]$ serves the role of a Skolem function. Either way, the value of $I.B$ is known to be some other frame K, so that $I.B.\tau$ is equal to $K.\tau$. We make $K.\tau$ a parent of $I.B.\tau$, and define the CPT to enforce this equality. These intermediate nodes along the slot chain are introduced to monitor the flow of values through complex slots. They are needed because the flow becomes complicated when the chain contains indirect slots. Intermediate variables that are spurious can easily be eliminated in a simple post-processing phase.

If B is indirect, then the value of $I.B$ could be one of several frames, depending on the value of the reference slot $I.\mathit{ref}(B)$. For any value ρ of $I.\mathit{ref}(B)$, let $K[\rho]$ denote the frame which is the value of $I.B$. The value of $I.B.\sigma$ is equal to the value of $K[\rho].\sigma$. In other words, $I.\mathit{ref}(B)$ selects the value of $I.B.\sigma$ from a set of possibilities. Therefore, the node $I.B.\sigma$ is a *multiplexer* node [1]; it has as parents the node $I.\mathit{ref}(B)$ and all nodes $K[\rho].B.\sigma$, and it uses the value of $I.\mathit{ref}(B)$ to select, as its value, the value of one of its appropriate parents.

ProcessIndirect($I.B.\tau$)
AddParent($I.B.\tau$, $I.\mathit{ref}(B)$)
For each value ρ of $I.\mathit{ref}(B)$
If ρ is a named individual J then $K[\rho] = J$
If ρ is the slot chain σ then $K[\rho] = I.\sigma$
If ρ is the class Y then $K[\rho] = Y[I.C]$
AddParent($I.B.\tau$, $K[\rho].\tau$)
Set CPT for $I.B.\tau$ to select the value of $K[I.B].\tau$

It remains to deal with the cases introduced by number uncertainty. Since a multi-valued slot A can only be used by quantifier slots, these are the only slots which we need to consider. Consider a quantifier slot $I.Q$ over $A.\sigma$. The value of $I.Q$ is fully determined by $I.\mathit{num}(A)$ and the value of σ in each of the possible values of A. Let n be the maximum number of such values, and suppose that A is assigned m values $J[1], \ldots, J[m]$ in I. In addition to these, there can be up to $n - m$ other instances that are values for A; we build a frame for each of them, $J[m+1], \ldots, J[n]$. The node $I.Q$ depends on $I.\mathit{num}(A)$ and on the appropriate subset of the variables $J[i].\sigma$; i.e., if $\mathit{num}(A)$ is k, then only $J[1], \ldots, J[k]$ will influence $I.Q$. The exact form of the dependence will depend on the form of the quantifier. For example, a $\forall$ quantifier slot will be a deterministic conjunction of the appropriate subset of its parents.

ProcessQuantifier($I.Q$) /* a quantifier over $A.\sigma$ */
AddParent($I.Q$, $I.\mathit{num}(A)$)
Let n be the maximum value of $I.\mathit{num}(A)$
Let $J[1], \ldots, J[m]$ be the assigned values to $I.A$
For $i = m + 1$ to n
$J[i] = X[i][I.A]$, where X is the type of $I.A$
For i = 1 to n
AddParent($I.Q$, $J[i].\sigma$)
Set the CPT for $I.Q$ to depend on $J[1].\rho, \ldots, J[\mathit{num}(A)].\sigma$

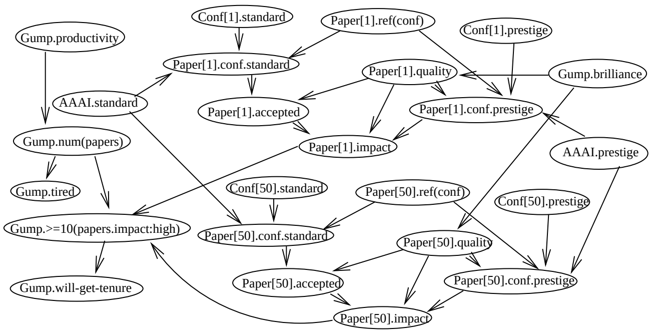

Figure 2: Part of the constructed BN for Prof. Gump's tenure case. Models for only two of the fifty possible papers are shown. Paper[i] is shorthand for paper[Gump.*papers*][i], and Conf[i] is short for conf[paper[Gump.*papers.conference*][i]].

To illustrate the algorithm, Figure 2 shows part of the BN constructed for a KB concerning the tenure case of one Prof. F. Gump, an instance of ai-assistant-professor. The node Gump.*will-get-tenure* depends on Gump.$\geq$ 10(*papers.impact:high*). The latter is a quantifier slot, so it has as parents Gump.$\mathit{num}(\mathit{papers})$ and the *impact* slot of each of the 50 possible papers. Now, assume that the class paper has *impact* depending on *conference.prestige*. This dependence is duplicated for each of the 50 possible papers. The slot *conference* is indirect, so that for each i (using the shorthand of the figure) *Paper[i].conf.prestige* has the parent *Paper[i].ref (conf)* and the *prestige* slot of the possible values of that reference, which are AAAI and *conf*[i].

Looking over the overall structure of the algorithm, we see that there is a correspondence between the structure of the dependency graph $\mathcal{G}$ and that of the BN $\mathcal{B}$. We can define a mapping φ from nodes of $\mathcal{B}$ to nodes of $\mathcal{G}$ as follows: $\varphi(I.A.\sigma)$ is $I.A$ if I is a named individual, and $X.A$ otherwise for X the p-class of I. Intuitively, the node $X.A$ in $\mathcal{G}$ is a representative for all the nodes $I.A$ where I is a generic instance of X.

Lemma 6.1: *If there is an edge from node μ_1 to node μ_2 in $\mathcal{B}$, then there is an path from $\varphi(\mu_1)$ to $\varphi(\mu_2)$ in $\mathcal{G}$.*

In other words, $\mathcal{G}$ serves as a template for the dependencies in $\mathcal{B}$. Many edges in $\mathcal{B}$ may map to the same path in $\mathcal{G}$, but any cycle or infinite dependency in $\mathcal{B}$ will necessarily map to a cycle in $\mathcal{G}$ (because $\mathcal{G}$ is finite). This property is the basis for the following theorem.

Theorem 6.2: *If the dependency graph is acyclic,* **ConstructBN** *terminates and the constructed Bayes net is acyclic.*

This construction also provides us with an alternative specification for the distribution defined by a probabilistic frame KB. Intuitively, the BN corresponds to the prior distribution defined by the KB. In particular, the CPT of a $\mathit{num}(A)$ slot can ascribe a positive probability to $\mathit{num}(A) = 0$, despite the fact that one or more values have been asserted for A in the

KB. In order for $\mathcal{B}$ to represent the distribution defined by our semantics, we must condition it on all of our observations. Specifically, we assert the value for any simple slot whose value was assigned in the KB, and lower bounds on the value of *num*(A) corresponding to the number of values asserted for A (including indirectly via inverses).

Theorem 6.3: *Let $\mathcal{S}$ be the set of simple slots of named individuals, and $\mathcal{E}$ the evidence on simple slots and number slots derived from the KB. Then $\Pr_{\mathcal{B}}(\mathcal{S} \mid \mathcal{E})$ is the same as the distribution over $\mathcal{S}$ defined by our semantics.*

We have implemented our approach within a system that contains the following functionality: A graphical network-based editor/browser can be used to annotate a frame KB with the facets encoding its probability model. The editor/browser interacts with the underlying FRS using OKBC, thus allowing its use with many of the existing frame systems (e.g., [3, 8, 7]). Our inference component also connects to the FRS via OKBC; it extracts the relevant information about frames, instances, and their probabilistic models, constructs the BN corresponding to the KB, and utilizes the BN to answer probabilistic queries. The system has been integrated successfully with the Ontolingua frame system [3] and was used for representing simple models of vehicle movement patterns in a military setting. Our experience with the system showed that even very simple models with three or four simple p-classes could be used to generate fairly complicated BNs (with hundreds of nodes) involving several interacting entities.

7 Discussion and conclusions

In this paper, we have described the first integration between two of the most dominant threads of work in KR. Our language provides a clean synthesis between the probabilistic reasoning component and standard frame reasoning capabilities. From the perspective of frame systems, our system allows existing frame KBs to be annotated with probabilistic models, greatly increasing the ability of frame systems to express meaningful knowledge in real-world applications. We have also provided an inference algorithm capable of answering probabilistic queries about instances, thereby providing a significant increase to the inferential ability of such systems. From the perspective of probabilistic modeling, our language provides the tools for the construction of probabilistic models for very large complex domains, significantly scaling up our ability to do uncertain reasoning.

Our language has given us the exciting capability of creating highly expressive probabilistic models with structural uncertainty. Clearly, we have only scratched the surface of this idea. For example, it is easy to add uncertainty over the type of an object, e.g., to define a probability distribution with which a professor is an assistant, associate, and full professor. It is also easy to combine number and reference uncertainty, allowing, for example, the advisor of a student to be selected from the set of faculty members in the CS department. These are two of many possible extensions that can now be considered.

Another important issue which we have partially resolved is the inference problem for probabilistic frame-based models. We have shown how we can reduce the problem to that of reasoning in a standard BN, but this approach does not make full use of the structure encoded in our representation. In particular, it fails to exploit encapsulation of frames within other frames and the reuse of class models among several objects. These ideas are put to good use in [6, 5], and it is an important research topic to apply them in our richer framework. We believe that by exploiting these features in our inference as well as in our representation, we will be able to effectively represent and reason in large uncertain domains.

Acknowledgments This work was supported by ONR contract N66001-97-C-8554 under DARPA's HPKB program, by DARPA contract DACA76-93-C-0025 under subcontract to Information Extraction and Transport, Inc., and through the generosity of the Powell Foundation and the Sloan Foundation.

References

[1] C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-specific independence in Bayesian networks. In *Proc. UAI*, 1996.

[2] V.K. Chaudhri, A. Farquhar, R. Fikes, P.D. Karp, and J.P. Rice. Open knowledge base connectivity 2.0.2. Available from `http://www.ai.sri.com/~okbc`, 1998.

[3] A. Farquhar, R. Fikes, and J. Rice. The Ontolingua server: A tool for collaborative ontology construction. Technical report, Stanford KSL 96-26, 1996.

[4] M. Henrion. Propagation of uncertainty in Bayesian networks by probabilistic logic sampling. In *Proc. UAI*, 1988.

[5] D. Koller, A. Levy, and A. Pfeffer. P-classic: A tractable probabilistic description logic. In *Proc. AAAI*, 1997.

[6] D. Koller and A. Pfeffer. Object-oriented Bayesian networks. In *Proc. UAI*, 1997.

[7] D.B. Lenat and R.V. Guha. *Building Large Knowledge-Based Systems: Representation and Inference in the CYC Project*. Addison-Wesley, 1990.

[8] R. MacGregor. The evolving technology of classification-based knowledge representation systems. In J. Sowa, editor, *Principles of semantic networks*, pages 385–400. Morgan Kaufmann, 1991.

[9] L. Ngo and P. Haddawy. Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 1996.

[10] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

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

[12] M.P. Wellman, J.S. Breese, and R.P. Goldman. From knowledge bases to decision models. *The Knowledge Engineering Review*, 7(1):35–53, November 1992.