Mach Learn (2006) 62: 7–31
DOI 10.1007/s10994-006-5831-3

# PRL: A probabilistic relational language

**Lise Getoor · John Grant**

Received: 17 August 2004 / Revised: 24 June 2005 / Accepted: 1 September 2005 /
Published online: 27 January 2006


**Abstract** In this paper, we describe the syntax and semantics for a probabilistic relational language (PRL). PRL is a recasting of recent work in Probabilistic Relational Models (PRMs) into a logic programming framework. We show how to represent varying degrees of complexity in the semantics including attribute uncertainty, structural uncertainty and identity uncertainty. Our approach is similar in spirit to the work in Bayesian Logic Programs (BLPs), and Logical Bayesian Networks (LBNs). However, surprisingly, there are still some important differences in the resulting formalism; for example, we introduce a general notion of aggregates based on the PRM approaches. One of our contributions is that we show how to support richer forms of structural uncertainty in a probabilistic logical language than have been previously described. Our goal in this work is to present a unifying framework that supports all of the types of relational uncertainty yet is based on logic programming formalisms. We also believe that it facilitates understanding the relationship between the frame-based approaches and alternate logic programming approaches, and allows greater transfer of ideas between them.

**Keywords** Probabilistic relational models · Logic programming · Uncertainty

## 1. Introduction

There is a long tradition of attempts to understand the semantics of a language which marries first-order logic with probability theory (Gaifman, 1964; Halpern, 1990; Michael Wellman

**Editors:** Hendrik Blockeel, David Jensen and Stefan Kramer

L. Getoor (✉)
Department of Computer Science and UMIACS, University of Maryland,
College Park, MD, 20742
e-mail: getoor@cs.umd.edu

J. Grant
Department of Computer and Information Sciences and Department of Mathematics, Towson University,
Towson, MD 21252
e-mail: jgrant@towson.edu

& Goldman, 1992). In the 1990s, several first-order probabilistic languages were developed (Poole, 1993; Jaeger, 1998; Koller & Pfeffer, 1998; Kersting & De Raedt, 2001a). The majority of these can either be categorized as logic-programming based (Muggleton, 1996; Sato and Kameya, 1997; Kersting & De Raedt, 2001b; Costa et al., 2003) or based on frame-based or entity-relationship models (Koller & Pfeffer, 1998; Heckerman et al., 2004). Many can be seen as attempts to extend Bayesian networks (Pearl, 1988) to relational and first-order domains (Koller & Pfeffer, 1998; Jaeger, 1998; Heckerman et al., 2004) while others can be seen as attempts to augment logic programming (Lloyd, 1989) with uncertainty (Muggleton, 1996; Sato & Kameya, 1997; Kersting & De Raedt, 2001a).

Probabilistic relational models (PRMs) (Koller & Pfeffer, 1998) are representative of the family of approaches based on frame-based formalisms. PRMs extend directed graphical models to relational domains. An advantage of PRMs is that they allow the types of generalizations common in first-order and object-oriented systems while at the same time providing a compact probabilistic model. Efficient inference techniques have been developed for PRMs (Koller et al., 1997; Pfeffer et al., 1999; Pfeffer, 2000; Pasula & Russell, 2001) and learning techniques have been developed (Friedman et al., 1999; Getoor et al., 2001a; Getoor, 2001). Techniques for learning models in which there is relational uncertainty have also been developed (Getoor et al., 2001b; 2002). There have been applications of PRMs to a number of problems including the study of tuberculosis epidemiology in medical domains (Getoor et al., 2004), collective classification in bibliographic and web collections (Getoor et al., 2001c) and to the problem of selectivity estimation in databases (Getoor et al., 2001d).

Here we present a logic programming approach to probabilistic relational models. We show how both logical and probabilistic structure can be captured and show how a variety of probabilistic models of structural uncertainty can be supported. Our approach is in the spirit of Kersting and De Raedt (2001b) and (Fierens et al., 2004, 2005). As in the case of Fierens et al. and Blockeel (2003), our approach attempts to make a clear distinction between logical structure and probabilistic dependency structure. In addition, our approach supports aggregation and allows uncertainty over the logical structure. We believe that the main ideas in our approach to aggregation and structural uncertainty are relatively general, and could be used to extend the other systems as well. We call our language PRL (pronounced 'pearl') for Probabilistic Relational Language, and introduce the notion of a PRL program.

Our original goal in this work was to recast PRMs in a logic programming framework. We hoped that this would help to illustrate commonalities and differences in the frame-based and logic programming approaches. PRMs put an emphasis on the connections to relational databases and frame-based systems. Here we have translated that to a logic programming approach. The resulting language is closely related to existing probabilistic logic programming approaches such as BLPs (Kersting & De Raedt, 2001b) and LBNs (Fierens et al., 2004). We believe by doing this, and then showing how to import some novel ideas from PRMs, for example the definition of aggregates and the different forms of structural uncertainty, we have both illustrated the commonality in the underlying approaches and contributed to the development of the general line of work in probabilistic logic programming languages.

The format of this paper is as follows. In Section 2 we give an intuitive example that we use throughout the paper to illustrate concepts. Section 3 contains a description of the syntax for PRL. Section 4 explains the semantics. Section 5 shows how various types of structural uncertainties are handled in PRLs. In Section 6 we briefly compare PRLs with BLPs and LBNs. The paper is concluded in Section 7.

## 2. An example

We use a simple and hopefully intuitive running example to illustrate the syntax and semantics for our language. The example is in the spirit of the professor, student, courses example introduced in Getoor et al. (2001a), but, as we shall see, allows some more complex probabilistic relationships. The example involves researchers, institutions, and publications; we'll call this example the research world domain (or the RW example for short).

In this example, we will represent entities such as researchers, institutions and papers and relationships such as author-of and cites. Entities may have attributes, for example a researcher is described by his or her research area and salary. An institution may be an educational institution, a research lab or an industrial institution. Papers have associated topics. Some of the relationships in this domain include: authorship, i.e., researchers publish papers, and citation, i.e., papers cite other papers. This example can be extended in many interesting ways, for example by including venue information for papers, information about grants and funding, and information about students. We will expand the example as needed as we go along.

Each of the entities and relationships in this example is represented by a predicate. It will be useful to distinguish the logical structure of the domain, which will be represented using primary keys and foreign keys, from the probabilistic aspects of the domain, which will be represented as a distribution over the non-key attributes in the domain. In our case, we have the following set of predicates:

- *Institute*(*InstId*, *Type*)
  The primary key is InstId. Type is a non-key attribute.
- *Researcher*(*RId*, *Area*, *Salary*, *InstId*)
  The primary key is RId. InstId is a foreign key that references InstId in Institute. Area and Salary are non-key attributes.
- *Paper*(*PId*, *Topic*)
  The primary key is PId. Topic is a non-key attribute.
- *Author*(*RId*, *PId*)
  The primary key is RId and PId. RId is a foreign key that references Researcher and PId is a foreign key that references Paper.
- *Cites*(*PId*1, *PId*2)
  The primary key is (PId1, PId2). Both PId1 and PId2 are foreign keys that reference Paper.

Starting with the next section, we will show how to represent the following kinds of dependencies:

**intra-relational dependencies** - e.g., A researcher's salary depends on their research area.

**inter-relational dependencies** - e.g., A researcher's salary depends on the type of institute at which they work.

**reference uncertainty** - e.g., A paper's author is more likely to be a researcher in the same area as the paper.

**exists uncertainty** - e.g., A citation between two papers is more likely to exist if they are on the same topic.

**identity uncertainty** - e.g., The authors of two distinct papers are more likely to be the same individual if the author names are similar and if the co-authors are the same.

## 3. Basic concepts and syntax

We generally use a standard logic programming formalism (Lloyd, 1989). Here we explain the main concepts and special notation appropriate for our purpose. We start by presenting the basic concepts of the syntax and will expand on various aspects of it in later subsections. We describe both the necessary elementary logical notions and probabilistic representations. Once all of these are in place, in Section 4, we describe the semantics.

Intuitively a PRL program has three components: 1) a logical background theory which defines the set of variables used to describe the logical structure of the domain, 2) a probabilistic background theory which, together with the logical background theory, defines the set of random variables and 3) a probabilistic dependency theory which, together with the logical theory and probabilistic background theory, defines a probability distribution.

A PRL is defined over a typed first-order language. Each variable and constant symbol is associated with a type $\tau$. Each type $\tau$ has an associated domain $\mathrm{dom}(\tau)$. Constant symbols are represented by a string beginning with a lowercase letter or with a number. Variables are represented by a string beginning with an uppercase letter. We use the shorthand $\mathrm{dom}(X)$ to denote the domain of a variable $X$. We assume that all domains are finite, although this restriction can be lifted in some cases. Predicate symbols have various arities, and their arguments are typed. For each PRL program, there is a set of *basic* predicates. For these predicates, the arguments are designated as either key, foreign-key or non-key. The distinction among these will be described in the next subsection. Predicates are typically represented by a string beginning with an uppercase letter, followed by a list of arguments. Function symbols also have various arities and their arguments and return values are typed. Function symbols are typically represented by a string beginning with a lowercase letter, followed by a list of arguments.

### 3.1. Logical background theory

A logical theory is used to describe the universe of discourse for a PRL. The logical theory describes the key constraints and includes a background set of facts.

#### *3.1.1. Keys*

For each basic predicate describing a relation in our domain, we designate (outside of the logic) an argument (or set of arguments) in each predicate as the primary key. The meaning of this concept can be represented by formulas, standing for constraints, in the language. For example, if the basic predicate is

$$\mathit{Researcher}(\mathit{RId}, \mathit{Area}, \mathit{Salary}, \mathit{InstId})$$

and *RId* is the primary key, the formulas for the functional dependency (key constraint) are:

$$\begin{aligned}
\mathit{Area}1 = \mathit{Area}2 \leftarrow\ & \mathit{Researcher}(\mathit{RId}, \mathit{Area}1, \mathit{Salary}1, \mathit{InstId}1), \\
& \mathit{Researcher}(\mathit{RId}, \mathit{Area}2, \mathit{Salary}2, \mathit{InstId}2) \\
\mathit{Salary}1 = \mathit{Salary}2 \leftarrow\ & \mathit{Researcher}(\mathit{RId}, \mathit{Area}1, \mathit{Salary}1, \mathit{InstId}1), \\
& \mathit{Researcher}(\mathit{RId}, \mathit{Area}2, \mathit{Salary}2, \mathit{InstId}2)
\end{aligned}$$

$$InstId1 = InstId2 \leftarrow Researcher(RId, Area1, Salary1, InstId1),$$
$$Researcher(RId, Area2, Salary2, InstId2)$$

In many cases the primary key is a single argument; by convention we usually make it the first argument of the relation. However we may also have a set of arguments as keys; for example the key for the *Author*(*RId*, *PId*) predicate is {*RId, PId*}.

For each basic predicate *P(KId,* $X_1, \ldots, X_n$*)*, we introduce an auxiliary inst predicate ('inst' connoting instantiation), *PInst*(*KId*), where *KId* is the primary key. For example, given

*Researcher*(*RId*, *Area*, *Salary*, *InstId*),

we introduce *Researcher-Inst(RId).* We will refer to the set of these inst predicates as the *primary-key inst predicates*.

We also introduce the notion of a foreign key. Suppose that *InstId* is the primary key of a relation, *Institute*(*InstId, Type*), and we have the above predicate, *Researcher*(*RId, Area, Salary, InstId*). In the researcher predicate, we call *InstId* a foreign key because it references the primary key of *Institute*. The *InstId* argument of *Researcher* refers to institutes, and its domain is the same as the domain of the argument *InstId* of Institute. In other words, the foreign key in one predicate refers to the primary key of another predicate, and the domain of the foreign key is the same as the domain of the primary key. By convention, for both primary keys and foreign keys, we usually give arguments a name ending in "Id".

The foreign-key constraint for *InstId* can be represented by the formula:

$\exists$*Type Institute(InstId, Type)* $\leftarrow$ *Researcher(RId, Area, Salary, InstId)*

meaning that for every researcher, the institute of the researcher must appear in the relation Institute.

Generally we limit the language to definite clauses (we will be more precise about this shortly). However, the above clause is allowed in our language. Because the above clause is used strictly as a constraint, we can transform it to a Datalog clause as follows: Define a new predicate

*KnownInstitution*(*InstId*) $\leftarrow$ *Institute*(*InstId*, *Type*)

and write the constraint as the formula

$\leftarrow$ *Researcher*(*RId*, *Area*, *Salary*, *InstId*), $\neg$*KnownInstition*(*InstId*)

We will find the concept of foreign key useful for describing dependencies among random variables.

For each predicate *P*(*KId,* $X_1, \ldots,$ *FId*, $\ldots, X_n$), where *FId* is a foreign key with domain dom(*FId*), we introduce an auxiliary inst predicate *PFInst*(*KId, FId*), where *KId* is the primary key of relation *P* and *FId* is a foreign key of relation *P*. For example, given the basic predicate *Researcher*(*RId, Area, Salary, InstId*), we introduce *ResearcherInstitute-Inst*(*RId, InstId*). We will refer to the set of these predicates as the *foreign-key inst predicates*, and

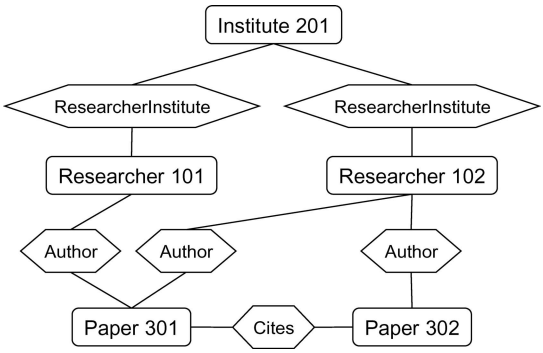


**Fig. 1.** A graphical representation for the logical background theory for a small RW example.

we will refer to the collection of both the primary-key inst predicates and foreign-key inst predicates as the set of *key inst predicates*.

### *3.1.2. Facts*

An essential component of any PRL program is a logical background theory describing the logical structure in the domain. There are a variety of constraints that we can put on the background theory to ensure that the semantics are well-defined. The first case we consider is the simplest, and corresponds to the "attribute-uncertainty" semantics from Friedman et al. (1999) and Getoor et al. (2002).

In this case, we require the background theory to provide ground extensional definitions for all of the primary-key and foreign-key predicates. We make a closed world assumption, and assume that any key predicates that do not hold in the background theory are false.

The following is an example of a set of facts for the RW domain:

*Researcher-Inst*(101)
*Researcher-Inst*(102)
*ResearcherInstitute-Inst*(101, 201)
*ResearcherInstitute-Inst*(102, 201)
*Institute-Inst*(201)
*Paper-Inst*(301)
*Paper-Inst*(302)
*Author-Inst*(101, 301)
*Author-Inst*(102, 301)
*Author-Inst*(102, 302)
*Cites-Inst*(301, 302)

Figure 1 shows a graphical representation of these facts.

As mentioned earlier, under the attribute uncertainty semantics, we make a closed world assumption which says that the logical background facts tell us all of the primary keys for each relation, and, further, that we know all of the logical relations defined by foreign keys. In the PRM work, this structure was called the *relational skeleton*. Note that, in turn, this defines the domains for the primary keys and foreign keys. So another way to think of this is that, at least in part, the logical background theory describes domain-closure constraints for the primary keys and foreign keys.

## 3.2. Probabilistic component

Next, we describe the mechanism for describing the *probabilistic* dependency structure of the domain. We refer the reader to DeGroot (1970) and Pearl (1988) for basic concepts of probability such as random variables and Bayesian networks. There are three components

here: 1) a probabilistic background theory, which, together with the logical background theory, defines the set of random variables in our domain, 2) a probabilistic dependency theory, which, together with the logical background theory and random variables, defines probabilistic dependencies among the random variables, and 3) a set of parameters, which defines the conditional probability for each random variable, given its parents in the probabilistic dependency graph.

### *3.2.1. Probabilistic background theory*

Once we have the logical background theory, we use this, together with a set of rules to define a set of random variables. These rules can be thought of as providing the random variable declarations. The random variables are introduced as Skolem functions of logical variables.

For each basic predicate, we have a rule which introduces the random variables for all the non-key attributes. For example for the basic predicate *Researcher*(*Rid*, *Area*, *Salary*, *InstId*), we have the following rule:

$$\exists \mathit{Area},\ \mathit{Salary}\ \mathit{Researcher}\ (\mathit{RId},\ \mathit{Area},\ \mathit{Salary},\ \mathit{InstId}) \leftarrow \mathit{Researcher\text{-}Inst}(\mathit{RId})$$

This introduces random variables *area*(*RId*), *salary*(*RId*) for each *Researcher-Inst*(*RId*) in the logical background theory. In the above example, the random variables introduced are: *area*(101), *salary*(101) and *area*(102), *salary*(102).

The rest of the rules for defining random variables for our example are:

$$\exists \mathit{Type}\ \mathit{Institute}(\mathit{InstId},\ \mathit{Type}) \leftarrow \mathit{Institute\text{-}Inst}(\mathit{InstId})$$

$$\exists \mathit{Topic}\ \mathit{Paper}(\mathit{PId},\ \mathit{Topic}) \leftarrow \mathit{Paper\text{-}Inst}(\mathit{PId})$$

$$\mathit{Cites}(\mathit{PId}1,\ \mathit{PId}2) \leftarrow \mathit{Cites\text{-}Inst}(\mathit{PId}1,\ \mathit{PId}2)$$

The intensional predicates above define the (unknown) random variables over which our distribution will be defined. We refer to this collection of rules as the *random variable declaration* rules. The last rule is not strictly needed, as there are no non-key random variables in the *Cites* predicate. In our example, the PRL program will define a joint distribution over:

$$\mathit{area}(101),\ \mathit{salary}(101),\ \mathit{area}(102),\ \mathit{salary}(102),\ \mathit{type}(201),\ \mathit{topic}(301),\ \mathit{topic}(302)$$

We'll denote this set of random variables by RV; these are the random variables defined by the random variable declarations in the probabilistic background theory.

### *3.2.2. Probabilistic dependency theory*

Next, we describe the mechanism for describing the *probabilistic* dependency structure of the domain. In this work, we introduce a special intentional predicate, DependsOn, which describes the *structure* of the probabilistic dependencies in the model. The concept of the dependency of a predicate argument on another argument or arguments of possibly different predicates is represented by a subscripted DependsOn predicate. For example, we may want

to say that a researcher's salary depends on the researcher's area. Perhaps researchers in 'hot' areas are paid more than researchers in more traditional areas. We would write this as follows:

$$\text{DependsOn}_1(salary(RId);\ area(RId)) \leftarrow Researcher\text{-}Inst(RId)$$

By convention, the first argument of any DependsOn predicate is the target random variable; its value depends probabilistically on the values of the remaining random variables. The meaning of the intentional axiom is that a researcher's salary depends on that person's research area. Note that this is a template for the dependencies. In our simple example, the above DependsOn translates into two dependencies: *salary*(101) on *area*(101) and *salary*(102) on *area*(102).

We may make this example more complex by saying that a researcher's salary depends on their area and on the type of institute at which they work. In that case we would write:

$$\text{DependsOn}_2(salary(RId);\ area(RId),\ type(InstId)) \leftarrow Researcher\text{-}Inst(RId),\ ResearcherInstitute\text{-}Inst(RId,\ InstId).$$

We need to include several rules about how the DependsOn predicate should be defined. We start with three general rules that are easy to state and follow with two more rules that we discuss further.

1. The set of DependsOn predicates are all intensional and occur only in the heads of clauses.
2. The body of a clause whose head is a DependsOn predicate may contain key inst predicates, built-in predicates, and negations of built-in predicates.
3. Every non-key argument $A$ of a basic predicate $R$ with key $K$ ($K \neq A$) must appear as the first argument in a DependsOn predicate.

Next we consider the case where there is more than one DependsOn predicate for a particular predicate argument. In this case we require that for each possible primary key of the target attribute, *exactly* one clause can be applied. For example, the following two clauses would be allowed:

$$\begin{aligned}&\text{DependsOn}_1(salary(RId);\ area(RId)) \leftarrow Researcher\text{-}Inst(RId),\\ &\qquad Area = \text{“}networking''\\ &\text{DependsOn}_2(salary(RId);\ area(RId),\ type(InstId)) \leftarrow ResearcherInst(RId),\\ &\qquad ResearcherInstitute\text{-}Inst(RId, InstId),\\ &\qquad Area \neq \text{“}networking''\end{aligned}$$

Another approach is to use combining functions as done in Kersting and De Raedt (2001a).

### *3.2.3. Aggregates*

In order to extend the expressivity of the language, we introduce the notion of aggregation. We allow the definition of a group of aggregation predicates that can be used in the bodies

of rules. These predicates can be used to capture the notion of aggregation in slot chains as described in Friedman et al. (1999), however they are significantly more expressive.

Suppose we change our example slightly, and want to say that a researcher's salary depends on the number of publications they have (or the number of grants). We can introduce an aggregate relation that counts the number of publications of an author:

$$Count_{RId}^{Author;PId}(RId,\ PaperCnt)$$

The meaning here is that we are defining an intentional predicate *Count*(*RId*, *PaperCnt*) which computes the count aggregate over the *PId* argument of *Author*(*RId*, *PId*). The key of the new predicate is *RId*. [1]

For example, if we have the following set of Authors: *Author*($a1$, $p1$), *Author*($a1$, $p2$), *Author*($a2$, $p1$), *Author*($a2$, $p3$), *Author*($a2$, $p4$), then the following aggregate predicates would hold:

$$Count_{RId}^{Author;PId}(a1,\ 2), \quad \text{and} \quad Count_{RId}^{Author;PId}(a2,\ 3).$$

The general form is

$$Aggr_{Key}^{Predicate;\ Aggr\text{-}Variable\text{-}List}(Key,\ AggrVal)$$

where the *Predicate* is the predicate over which the aggregate is computed, *Key* is a set of arguments that will form the key for the new aggregate relation, *AggrVal* is the value of the aggregate function applied to the set of values defined by *Aggr-variable-List*. Note that the predicate may be an intentional predicate as well; this will allow us to form quite complex aggregates over attributes from multiple relations.

For any aggregate predicate, we add two auxiliary predicates, analogous to the auxiliary predicates introduced for basic predicates. The first is a key predicate, *Aggr-Inst*(*Key*) . We include a simple rule

$$Aggr\text{-}Inst(Key) \leftarrow Predicate\text{-}Inst(Key).$$

The second is an aggregate variable declaration rule,

$$\exists AggrVal\, Aggr_{Key}^{Predicate;\ Aggr\text{-}Variable\text{-}List}(Key,\ AggrVal) \leftarrow Aggr\text{-}Inst(Key).$$

In our first example above, the key inst predicate introduced is *PaperCount-Inst* (*RId*) and the associated rule is

$$PaperCount\text{-}Inst(RId) \leftarrow Researcher\text{-}Inst(RId),$$

and the aggregate variable declaration rule is:

$$\exists PaperCntCount_{RId}^{Author;\ PId}(RId,\ PaperCnt) \leftarrow PaperCount\text{-}Inst(RId)$$

The above rule would introduce random variables *paper_cnt*(*RId*) for each author *RId*.

[1] The equivalent SQL for computing the aggregate would be:
select RId, count(PId) as PaperCnt
from Author group by Rid

Suppose that we want to compute a more complex aggregate, such as *mode*. As an example, suppose that, given a paper, we want to find the most common area of its authors. We define an intensional predicate *AuthorArea* which joins *Author* and *Researcher* and includes the variables we want to aggregate over:

$$\mathit{AuthorArea}(\mathit{PId}, \mathit{RId}, \mathit{Area}) \leftarrow \mathit{Author}(\mathit{RId}, \mathit{PId}),$$
$$\mathit{Researcher}(\mathit{RId}, \mathit{Area}, \mathit{Salary}, \mathit{InstId})$$

and define an aggregate to find the most common research area of the authors for each paper:

$$\mathit{Mode}_{\mathit{PId}}^{\mathit{AuthorArea};\mathit{Area}}(\mathit{PId}, \mathit{AreaMode})$$[2]

Next, we define an aggregate inst predicate:

$$\mathit{AuthorArea\text{-}Inst}(\mathit{PId}) \leftarrow \mathit{Author\text{-}Inst}(\mathit{RId}, \mathit{PId}),$$

and finally, we have a random variable declaration rule:

$$\exists \mathit{AreaMode}\, \mathit{Mode}_{\mathit{PId}}^{\mathit{AuthorArea};\mathit{Area}}(\mathit{PId}, \mathit{AreaMode}) \leftarrow \mathit{AuthorArea\text{-}Inst}(\mathit{PId})$$

The above rule would introduce a random variable *area_mode*(*PId*) for each *PId*.

### 3.3. Probabilistic parameters

So far we have not indicated how to specify the actual parameters of the distribution, or the probabilities. Each DependsOn predicate, $\textsc{DependsOn}_i(a_1; a_2, \ldots, a_n)$ has an associated conditional probability distribution $\textsc{cpd}_i(a_1 | a_2, \ldots, a_n)$. Let $v_1, \ldots, v_n$ denote a particular instantiation of $a_1, \ldots, a_n$. The CPD specifies, for each potential instantiation, a probability that is between 0 and 1 and, for any given instantiation of $a_2, \ldots, a_n$,

$$\sum_{v_1 \in \mathrm{dom}(a_1)} \mathrm{CPD}_i(v_1 \mid v_2, \ldots, v_n) = 1$$

Note that we allow arbitrary functional representations for the CPD. It is common to represent the CPD as either a table or a tree, or a set of rules, or other specific forms, such as noisy-or, noisy-max and so on. Each form has trade-offs in terms of number of parameters, expressive power and ease of use in inference and learning.

Completing our RW example, we have the following DependsOn predicates and CPD functions in our sample PRL program:

1. The institute type does not depend on any attributes. Suppose the possible institute types are research (res) and teaching (tch).

$$\textsc{DependsOn}_1(\mathit{type}(\mathit{InstId})) \leftarrow \mathit{Institute\text{-}Inst}(\mathit{InstId})$$
$$\textsc{cpd}_1(\mathit{res}) = 0.3,\ \textsc{cpd}_1(\mathit{tch}) = 0.7$$

[2] The equivalent SQL for computing the aggregate would be:
select PId, mode(Area) as AreaMode from AuthorArea
group by PId

2. The researcher's area depends on institute type. Suppose the possible research areas are software (sw), theory (th) and hardware (hw) and research universities have more theorists.

$$\textsc{DependsOn}_2(\mathit{area}(\mathit{RId}); \mathit{type}(\mathit{RId})) \leftarrow \mathit{Researcher\text{-}Inst}(RId), \mathit{ResearcherInstitute\text{-}Inst}(\mathit{RId}, \mathit{InstId})$$

$$\textsc{cpd}_2(sw \mid res) = 0.3,\ \textsc{cpd}_2(th \mid res) = 0.4,\ \textsc{cpd}_2(hw \mid res) = 0.3,$$

$$\textsc{cpd}_2(sw \mid tch) = 0.5,\ \textsc{cpd}_2(th \mid tch) = 0.1,\ \textsc{cpd}_2(hw \mid tch) = 0.4,$$

3. A researcher's salary depends on the number of papers authored. For simplicity, we assume that the number of papers is discretized into bins, few (f), many (m) and lots (l), and likewise salary is categorized as low (l), medium (m) and high (h).

$$\textsc{DependsOn}_3(\mathit{salary}(\mathit{RId}); \mathit{paper\_cnt}(RId)) \leftarrow \mathit{Researcher\text{-}Inst(RId)}, \mathit{PaperCount\text{-}Inst}(\mathit{RId})$$

$$\textsc{cpd}_3(l \mid f) = 0.6,\ \textsc{cpd}_3(m \mid f) = 0.3,\ \textsc{cpd}_3(h \mid f) = 0.1,$$

$$\textsc{cpd}_3(l \mid m) = 0.3,\ \textsc{cpd}_3(m \mid m) = 0.5,\ \textsc{cpd}_3(h \mid m) = 0.2,$$

$$\textsc{cpd}_3(l \mid l) = 0.2,\ \textsc{cpd}_3(m \mid l) = 0.5,\ \textsc{cpd}_3(h \mid l) = 0.3,$$

4. The topic of a paper depends on the most common area of the authors. We use the aggregate we defined earlier and then the DependsOn is:

$$\textsc{DependsOn}_4(\mathit{topic}(PId); \mathit{area\_mode}(PId)) \leftarrow \mathit{Paper\text{-}Inst}(PId), \mathit{AuthorArea\text{-}Inst}(PId)$$

$$\textsc{cpd}_4(sw \mid sw) = 0.90,\ \textsc{cpd}_4(th \mid sw) = 0.08,\ \textsc{cpd}_4(hw \mid sw) = 0.02,$$

$$\textsc{cpd}_4(sw \mid th) = 0.01,\ \textsc{cpd}_4(th \mid th) = 0.96,\ \textsc{cpd}_4(hw \mid th) = 0.03,$$

$$\textsc{cpd}_4(sw \mid hw) = 0.02,\ \textsc{cpd}_4(th \mid hw) = 0.03,\ \textsc{cpd}_4(hw \mid hw) = 0.95.$$

### 3.4. Summary of syntax

We have described the key components of a PRL program: 1) a logical background theory, which includes a collection of key predicate facts and 2) a probabilistic theory, which includes a collection of random variable declaration rules, a set of dependency clauses and a set of parameters, describing the conditional probability distributions. Other definitions can be added to the theory, for example definitions of aggregates and definitions of additional intensional predicates.

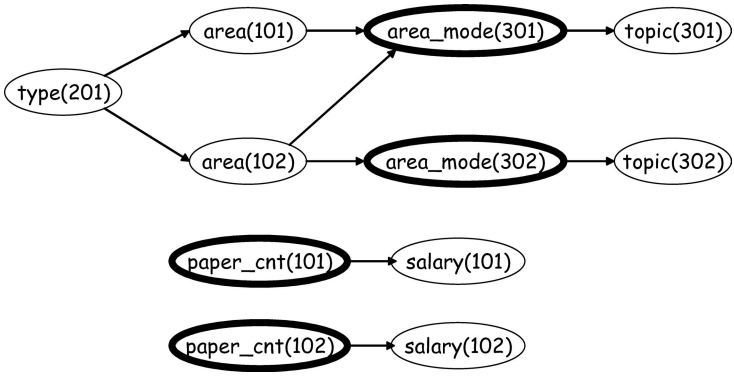


**Fig. 2.** The dependency graph for our running example. The darkly outlined ovals correspond to deterministic aggregate nodes, the other nodes correspond to random variables in the distribution.

## 4. Semantics

A PRL program defines a distribution over possible worlds. The possible worlds are complete instantiations of some set of random variables. We consider several different conditions on the background theory so that we can guarantee coherent probabilistic semantics. In the simplest case, the logical background theory provides complete information about the relational structure in the domain, and this, together with the DependsOn statements and CPDs, and possibly other intentionally defined aggregates and relations, defines the distribution. We use the notation DependsOn($V$; Parents($V$)) to denote the DependsOn associated with $V$, and CPD ($V$ | Parents($V$)) to denote the CPD associated with $V$.

The DependsOn predicates are the key to defining the distribution. We can use them to define a Bayesian network over the random variables. The dependency graph for the Bayesian network is a graph over the random variables RV. For any binding of a variable $V_1$ $\epsilon$ RV, where $V_1$ is a binding for $A_1$, which satisfies a DependsOn predicate DependsOn($A_1$; $A_2, \ldots, A_n$), we add edges from each binding $V_i$ in $A_2, \ldots A_n$ to $V_1$. For example consider the set of DependsOn examples 1-4 from the PRL program in the last section. We start with the DependsOn$_2$ from before:

$$\text{DependsOn}_2(\textit{area(RId)};\ \textit{type}(\textit{InstId})) \leftarrow \textit{Researcher-Inst}(\textit{RId}),\ \textit{ResearcherInstitute-Inst}(\textit{RId}, \textit{InstId})$$

There are two bindings for this rule, one for $RId = 101$ and $InstId = 201$, and another one for $RId = 102$ and $InstId = 201$. For the first binding, we would add an edge $type(201) \rightarrow area(101)$ and for the second binding, we would add an edge $type(201) \rightarrow area(102)$.

$V_i$ may also be an aggregate. In that case, we add edges between $V_1$ and the variables the aggregate is computed over. For example, given the

$$\textit{Count}_{RId}^{Author;PId}(\textit{RId},\ \textit{PaperCnt})$$

aggregate, and our earlier dependence of salary on number of publications:

$$\text{DependsOn}_3(\textit{salary}(\textit{RId}); \textit{paper\_cnt}(\textit{RId})) \leftarrow \textit{Researcher-Inst}(\textit{RId}),\ \textit{PaperCount-Inst}(\textit{RId})$$

then for each instantiation of the body, or for each *RId*, we add an edge from the aggregate *paper_cnt*(*RId*) to *salary*(*RId*). Because in this case the aggregate is the simplest aggregate count, the values are determined by the logical background theory. So the *paper_cnt*(*RId*) values are fixed, and we denote this with the double circle.

For other aggregates, that depend on attribute values, not just counts, we add edges from the attributes to the aggregate random variables. Consider the DependsOn for topic:

$$\text{DependsOn}_4(topic(PId); area\_mode(PId)) \leftarrow Paper\text{-}Inst(PId),\ AuthorArea\text{-}Inst(PId)$$

In this case, we add edges from author areas *area*(*RId*) to the *area_mode*(*PId*), and from *area_mode*(*PId*) to *topic*(*PId*). Paper 301 has authors 101 and 102, so we add edges from *area*(101) and *area* (102) to *area_mode*(301), and we add an edge from *area_mode*(301) to *topic*(301). Paper 302 has one author, researcher 102, so we add an edge from *area* (102) to *area_mode*(302) and we add an edge from *area_mode*(302) to *topic*(302). Again, the aggregate variable is deterministic, so we denote this with a double circle.

Figure 2 shows a sample dependency graph for our domain. As we can see, the resulting dependency graph is acyclic. This is a requirement, and we will discuss how this is enforced in the next section. The graph, together with the CPDs defined earlier, defines a Bayesian network. In the original PRM work, this was called the ground Bayesian network or the unrolled Bayesian network. The Bayesian network can be used to compute the probability of any particular instantiation of the variables. For example, consider the following instantiation: $area(101) = hw$, $salary(101) = high$, $area(102) = hw$, $salary(102) = low$, $type(201) = res$, $topic(301) = hw$, and $topic(302) = hw$. The probability of this instantiation is simply:

$$\begin{aligned}
&P(area(101) = hw, salary(101) = high, area(102) = hw, salary(102) = low,\\
&\qquad type(201) = res, topic(301) = hw, topic(302) = hw)\\
&= P(type(201) = res) \cdot P(area(101) = hw \mid type(201) = res)\\
&\quad \cdot P(area(102) = hw \mid type(201) = res) \cdot P(salary(101) = high \mid paper_cnt(101) = few)\\
&\quad \cdot P(salary(102) = low \mid paper_cnt(102) = few) \cdot P(topic(301) = hw\\
&\quad \mid area_mode(101) = hw) \cdot P(topic(302) = hw \mid area_Mode(302) = hw)\\
&= 0.3 \cdot 0.3 \cdot 0.3 \cdot 0.1 \cdot 0.6 \cdot 0.95 \cdot 0.95\\
&= 0.001465
\end{aligned}$$

### 4.1. Guaranteeing coherence

Two requirements are needed for our probabilistic semantics to be coherent:

1. Exactly one DependsOn statement can hold for each random variable $V \in \text{RV}$;
2. The dependency graph is acyclic.

Then, as with PRMs, the distribution can be written in a factored form and the probability of any particular instantiation is simply the product of the CPDs associated with each random variable assignment. This is written as follows:

$$P(\text{RV}) = \prod_{v \in RV} \text{CPD}(v; \text{Parents}(v))$$

where $v$ is a particular value for each $V \in \text{RV}$.

To ensure that there is no circularity in the clauses defining an attribute's dependence on other sets of attributes we have several options. One way to achieve this is to do a stratification of attributes. Here we require that each attribute $A_1$ that occurs as the first argument in a DependsOn definition must be in a higher stratum than the attributes $A_2, \ldots, A_k$ occurring in the rest of the clause. However, this requirement is quite stringent and would not even work for many examples we might wish to represent.

For example, suppose we would like to say that the topic of a paper depends on the topic of a paper that it cites (for simplicity we assume that each paper cites a single paper; if, as is more likely, there is more than one citation, we can introduce an aggregate). We would like to say that:

$$\text{DependsOn}(\textit{topic}(\textit{PId1}); \textit{topic}(\textit{PId2})) \leftarrow \textit{Paper-Inst}(\textit{PId}1), \textit{Paper-Inst}(\textit{PId}2), \textit{Cites-Inst}(\textit{PId}1, \textit{PId}2)$$

This is legal, as long as the Cites relation is acyclic. In theory this makes sense because of the temporal ordering; papers only cite papers that have already been published.[3] In the language of Friedman et al. (1999), we would call *Cites* a guaranteed acyclic relation.

We can enforce a more general acyclicity constraint in our framework as follows. We begin by converting our numbered DependsOn statements using a generic binary dependency relation, $X \prec Y$.

For each binding of a clause

$$\text{DependsOn}(a_i(\textit{Id}1); \ldots, a_k(\textit{Id}2), \ldots) \leftarrow P1\text{-}\textit{Inst}(\textit{Id}1, \ldots), P2\text{-}\textit{Inst}(\textit{Id}2, \ldots), \ldots$$

where $V_i$ is a binding for $a_i(\textit{Id}1)$ and $V_k$ is a binding for $a_k(\textit{Id}2)$ we add $V_i \prec V_k$. Similarly, if $V_k$ is the result of an aggregation function, say

$$\textit{Aggr}^{<\textit{Predicate}>;<\textit{Aggr-Variable-List}>}_{<\textit{key}>}(<\textit{key}>, A_k)$$

we add $V_i \prec V_k$, and we also add $V_k \prec V_j$ for each binding of $a_j$ to $V_j$ in *Aggr-Variable-List*.

Then we can add the new intentional predicate $\prec^*$. We add the rules:

$$V_1 \prec^* V_2 \leftarrow V_1 \prec V_2$$

$$V_1 \prec^* V_3 \leftarrow V_1 \prec V_2,$$

$$V_2 \prec^* V_3$$

and add the constraint

$$\leftarrow V_1 \prec^* V_2, V_2 \prec^* V_1$$

[3] Unfortunately, in practice, this is often not the case. If you look at real-world bibliographic databases, they are often cyclic. In these cases, other methods can be used to break the cyclicity.

This guarantees that the definition of DependsOn is not circular, and in turn that the PRL program is legal. Note that this notion of acyclicity requirement is different from the one given in the original PRM work. In the original PRM semantics, the networks were required to be acyclic for *all* possible relational skeletons. In the semantics above, we are checking for each PRL program, whether the semantics are coherent. Either approach has its place. The approach given here is more flexible, but potentially more expensive to check.

## 5. Structural uncertainty

In the previous section we assumed that for each relation we had all the key values (primary and foreign) given in our logical background theory and the probabilities were associated with the other, non-key, attributes. The probabilistic dependencies described by the DependsOn predicates did not contain any random variables in the body of the rules. Next we turn to the case where we have some uncertainty over the logical relationships in our problem domain. We explore several possibilities. The models presented here are the logical analogues of the PRM structural uncertainty methods described in Getoor et al. (2002) and Getoor (2001).

Existing probabilistic logic languages allow limited forms of structural uncertainty at best. The notions of structural uncertainty we introduce here are quite a bit richer. First, as we will see, there is a much tighter integration of the probabilistic model and the logical background theory. In the case of reference uncertainty, the parameters of the probabilistic component depend on aspects of the logical background theory. An advantage of this is that the probabilistic component can be used with a variety of background theories and thus is more general than an approach which must enumerate the potential logical structures. A second way in which there is a tighter integration of the probabilistic model and the logical background theory is that we allow probabilistic dependencies to 'flow' along uncertain structural aspects of the models. However, if we allow this, we must be careful to ensure that the probabilistic semantics are still correct. We show how to do this for each type of structural uncertainty.

### 5.1. Reference uncertainty

First consider the case where the primary keys are described in the logical background theory, but some of the foreign key instantiations are not given in the facts. One approach to dealing with this is to incorporate this missing structural component into our probabilistic model. We make a closed world assumption, assuming that the range of the foreign-keys can be determined from the background theory, and introduce probability distributions over the foreign-key values. This approach to structural uncertainty is called *reference uncertainty*.

We introduce the approach by extending our RW example. We start by adding to the set of basic predicates. Suppose that papers are published in some set of venues. Suppose the possible venues are: Symposium on Theory of Computation (stoc), Foundations of Computer Science (focs), International Conference on Software Engineering (icse), Programming Language Design and Implementation (pldi), and International Symposium on Computer Architecture (isca). Venues have associated areas and are of different tiers. We can represent this by the following two new basic predicates:

*PublishedIn*(*PId*, *VenueId*)

*Venue*(*VenueId*, *Area*, *Tier*)

Further, let's assume that the logical background theory describes the set of venues:

$$\mathit{Venue\text{-}Inst}(\mathit{stoc}), \mathit{Venue\text{-}Inst}(\mathit{focs}), \mathit{Venue\text{-}Inst}(\mathit{icse}), \quad \mathit{Venue\text{-}Inst}(\mathit{pldi}), \mathit{Venue\text{-}Inst}(\mathit{isca})$$

and suppose, as before, we have two papers, paper 301 and paper 302.

Now suppose we are not given an extensional definition for the PublishedIn predicate, but instead we have the rule:

$$\exists\ \mathit{VenueId}\ \mathit{PublishedIn}(\mathit{PId}, \mathit{VenueId}) \leftarrow \mathit{Paper\text{-}Inst}(\mathit{PId})$$

in our set of random variable declarations. This introduces a random variable *venue_id*(*PId*) for each *PId*. In our example, it introduces the random variables *venue_id*(301) and *venue_id*(302). *VenueId* is a foreign-key, and as with any key it has a domain associated with it. In this case the domain is the set of venue primary keys.

Next, we need to specify a distribution over the venue ids. There are a number of ways of doing this. The interesting twist comes because the domain of the foreign-key depends on the domain of the primary key which is described by the background knowledge. Further, like other random variables, we want to allow the value of the random variable to be conditioned on the values of other random variables.

The simplest case is when the distribution is not conditioned on the values of other random variables. For example, in the case above, suppose the DependsOn for *VenueId* is:

$$\textsc{DependsOn}(\mathit{venue\_id}(\mathit{PId})) \leftarrow \mathit{Paper\text{-}Inst}(\mathit{PId})$$

The interpretation for this DependsOn statement is that the distribution over *VenueId*s is uniform, i.e. the value is chosen randomly from the domain of *VenueId*s. The CPD is this case is simply:

$$\mathit{cpd}(\mathit{venueid}) = \frac{1}{|\,\mathrm{dom}(\mathit{VenueId})\,|}.$$

A more interesting case is when we want to say that the VenueId depends probabilistically on other random variables. For example, we may wish to say that the paper's topic will influence the venue. In particular, venues may correspond to different research areas, theory (th), software (sw) and hardware (hw):

$$\mathit{Venue}(\mathit{stoc}, \mathit{th}), \quad \mathit{Venue}(\mathit{focs}, \mathit{th}), \quad \mathit{Venue}(\mathit{icse}, \mathit{sw}), \quad \mathit{Venue}(\mathit{pldi}, \mathit{sw}), \quad \mathit{Venue}(\mathit{isca}, \mathit{hw})$$

And, we want to say

$$\begin{aligned}\textsc{DependsOn}(&\mathit{venue\_id}(\mathit{PId}); \mathit{area}(\mathit{venue\_id}(\mathit{PId})); \mathit{topic}(\mathit{PId}))\\ &\leftarrow \mathit{PublishedIn}(\mathit{PId,venue\_id}(\mathit{PId})),\\ &\quad\ \mathit{Paper\text{-}Inst}(\mathit{PId})\end{aligned}$$

Note that this introduces a limited form of function nesting.

The general form for a DependsOn for a foreign key is

$$\mathit{DependsOn}(\mathit{variable}; \langle \mathit{Partition\text{-}Variable\text{-}List}\rangle\ ; \langle \mathit{parents}\rangle)$$

The first step is that we use the Partition-Variable-List to partition the instances of the primary relation into sets for each possible instantiation of the variables in the partition list. In our example, the partition is on the *Area* of the venue, so the venues would be partitioned into three sets:

$\pi_{area=th} = \{stoc, focs\}$,

$\pi_{area=sw} = \{icse, pldi\}$,

$\pi_{area=hw} = \{isca\}$

Once a partition is chosen for the variable, the foreign-key value is chosen uniformly at random from that set. So, if we choose the software partition, we would have probability 0.5 of choosing *icse* and 0.5 of choosing *pldi*. The last step is that, just as with the distributions of the non-key random variables, we can condition the partition choice based on the value of some variables. In our example, we can say that the partition depends on the topic of the paper. Thus our CPD will describe the probability of a partition, given the parent variables. Here is a sample CPD:

$\text{CPD}(sw \mid sw) = 0.92$, $\text{CPD}(th \mid sw) = 0.06$, $\text{CPD}(hw \mid sw) = 0.02$,

$\text{CPD}(sw \mid th) = 0.03$, $\text{CPD}(th \mid th) = 0.94$, $\text{CPD}(hw \mid th) = 0.03$,

$\text{CPD}(sw \mid hw) = 0.03$, $\text{CPD}(th \mid hw) = 0.03$, $\text{CPD}(hw \mid hw) = 0.94$

This says for example that, given that the paper is a hardware paper, the probability of the venue being from the hardware partition is 0.94. Since the only venue in the hardware area is *isca*, there is probability 0.94 the venue is *isca*. Using similar reasoning, there is probability 0.015 that the hardware paper appears in *icse* (probability 0.03 of being in a venue in the software area, and uniform probability among the two software venues).

The probabilistic semantics remains the same; we add the random variables for the foreign-keys, and compute the joint probability. The main additional requirement to guarantee coherence is to properly model the dependencies introduced in the dependency graph. The first set of dependencies are straight-forward. For a dependence statement:

DEPENDSON(*variable*; ⟨*Partition-Variable-List*⟩ ; ⟨*parents*⟩)

we require that both the parents *and* the variables that define the partition come before the variable in the variable ordering. Suppose we have the following facts:

*Paper-Inst*(101), *Paper-Inst*(102)

*Venue-Inst*(*stoc*), *Venue-Inst*(*focs*), *Venue-Inst*(*icse*), *Venue-Inst*(*pldi*), *Venue-Inst*(*isca*)

$\exists$*VenueId PublishedIn*(*PId, VenueId*) $\leftarrow$ *Paper-Inst*(*PId*)

Then for the *venue*(101), we would require:

*venue*(101) $\prec$ *topic*(101), *venue*(101) $\prec$ *area*(*stoc*),

*venue*(101) $\prec$ *area*(*focs*), *venue*(101) $\prec$ *area*(*icse*),

*venue*(101) $\prec$ *area*(*pldi*), *venue*(101) $\prec$ *area*(*isca*)

and similarly for *venue*(102). The reason for this is that we must know the areas of all the venues before selecting venue(101) and venue(102) because we must be able to compute the sizes of the Venue partition. For a more formal treatment, see (Getoor et al., 2002).

In addition, the set of dependencies introduced by any DependsOn statement which uses the probabilistic foreign key in its body introduces additional new dependencies. Thus reference uncertainty allows us to have much richer probabilistic dependence structures, but we must take care that any dependencies which are based on a probabilistic foreign key such as VenueId are placed in a higher stratum than any dependencies based on them.

Consider the following DependsOn, where we extend our example so that papers have impact ratings which depend on the conference-tier of the venue:

$$\text{DependsOn}(impact(PId); tier(venue\_id(PId)) \leftarrow Paper\text{-}Inst(PId), PublishedIn(PId, venue\_id(PId))$$

In this case, the *venue_id*(*PId*) must be determined before *impact*(*PId*) can be determined, because we need to know which venue *tier*(*venue_id*(*PId*)) to depend on, so we will add an ordering constraint:

$$impact(PId) \prec venue\_id(PId)$$

As another example, we may want to say that a researcher's area depends on the most frequent venue in which they publish. The following DependsOn captures this notion:

$$\text{DependsOn}(area(RId); venue\_area\_mode(RId)) \leftarrow Researcher\text{-}Inst(RId), VenueAreaMode\text{-}Inst(RId)$$

where the aggregate

$$Mode^{AuthorVenue;Area}_{RId}(RId, VenueAreaMode)$$

is defined over the following relationship:

$$AuthorVenue(RId, venue\_id(Pid), Area) \leftarrow Author(RId,PId), PublishedIn(PId, venue\_id(PId)), Venue(venue\_id(PId), Area).$$

In this case, *venue_id* must be determined before the *venue_area_mode* can be determined, and *venue_area_mode* must be determined before *area*, so we will add the ordering constraint:

$$venue\_area\_mode(RId) \prec venue\_id(PId)$$

for each venue and

$$area(RId) \prec venue\_area\_mode(RId).$$

### 5.2. Existence Uncertainty

A second type of structural uncertainty that we may wish to represent is *Existence Uncertainty*. This again will allow us to introduce probabilities over the logical structure, unlike, in the case of attribute uncertainty, where we had assumed the full logical structure was given as part of the background set of facts. In the reference uncertainty case, the background facts told us the foreign keys that existed, and we simply had a distribution over the values of the entities that participated in the relations. Now, in the existence uncertainty case, we allow a distribution over the existence of a relation.

For the uncertain relationship, we do this by introducing an additional boolean argument *exists* into the relation. Consider the *Cites* relation. We can model uncertainty in the Cites relation by introducing the new basic relation:

$$CitesExists(PId1, PId2, CExists)$$

The intended interpretation is that if the Exists argument is true, there is a citation between PId1 and PId2, and if it is false, there is no citation.

In order to introduce the random variable, we add the following rule to our background probabilistic theory:

$$\exists CExists\, CitesExists(PId1, PId2, CExists) \leftarrow PaperInst(PId1),\quad PaperInst(PId2)$$

This introduces a random variable $cexists(PId1, PId2)$ for each $PId1$ and $PId2$. To disallow self-citations, we may say the probability of *CExists* being true in the case where $PId1 = PId2$ is 0.0.

We also add the following rule to our logical background theory:

$$Cites(PId1, PId2) \leftarrow CitesExists(PId1, PId2, True)$$

Now, the *CExists* random variable needs to have a CPD associated with it. As an example, we may have the following depends:

$$\begin{aligned} \textsc{DependsOn}(&cexists(PId1, PId2); topic(Id1), topic(Id2)) \\ &\leftarrow CitesExists(PId, CitedId, CExists), \\ &\quad Paper\text{-}Inst(PId1), \\ &\quad Paper\text{-}Inst(PId2) \end{aligned}$$

and the following CPD:

To ensure the dependencies are acyclic, just as with any CPD, for the *cexists* CPD we will add $cexists(PId1, PId2) \prec topic(PId1)$ and $cexists(PId1, PId2) \prec topic(PId2)$. In addition, we need to ensure that in the case where we use the *Cites* relation in the body of a rule, we can determine whether the Cites exists before using it. Suppose we would like to say that the impact of a paper depends on the number of citations. We introduce an aggregate to compute the number of citations:

$$Count_{PId1}^{Cites;PId2}(PId1, PId2)$$

| *topic* (*PId1*) | *topic* (*PId2*) | *cexists* = *True* | *cexists* = *False* |
|---|---|---|---|
| *sw* | *sw* | 0.01 | 0.99 |
| *sw* | *th* | 0.001 | 0.999 |
| *sw* | *hw* | 0.001 | 0.999 |
| *th* | *sw* | 0.001 | 0.999 |
| *th* | *th* | 0.015 | 0.9985 |
| *th* | *hw* | 0.001 | 0.999 |
| *hw* | *sw* | 0.001 | 0.999 |
| *hw* | *th* | 0.001 | 0.999 |
| *hw* | *hw* | 0.02 | 0.998 |

Now, because the *Cites* relation will only hold for *CitesExists* with *CExists* = *True*, we will count only the actual citations.

There may be other cases where we want to use the uncertain relation in the body of the rule. As long as we add the restriction that the *CExists* argument is true, we will be able guarantee that the dependencies are acyclic and the semantics are correct. The probabilistic interpretation remains the same; we just include the random variables for *CExists* and compute the joint probability using the chain rule as before.

### 5.3. Identity uncertainty

The final type of structural uncertainty we address is identity uncertainty (Pfeffer, 2000; Russell, 2001; Pasula et al., 2003). This type of uncertainty allows us to model uncertainty over the identity of a reference.

This type of uncertainty comes up naturally in a number of problems. Deduplication (removing duplicates in a database table) and co-reference resolution (in natural language processing, figuring out when two phrases in the text refer to the same individual) (Pasula et al., 2003; McCallum & Wellner, 2003; Bhattacharya & Getoor, 2004) are widely studied problems that can be modeled using identity uncertainty. In these problems we are trying to discover cases such as when two entries on a mailing list are in fact referring to the same individual or household, or when the author references of two papers refer to the same researcher. Note that there may be differences in the references, "J. Smith" versus "John Smith", but we still want to be able to figure out that they refer to the same person.

The simplest case is when we are still operating in a closed world, where we know the universe of possible individuals. Then we can use a mechanism quite similar to our reference uncertainty approach to model our uncertainty about the references.

We can model this by distinguishing the references, or mentions, of an individual with the identity of an individual. In the author identification task, we may say that we have some set of author references:

*Author*(*RefId, PId*)

*RefersToIndividual*(*RefId*, *RId*)

We can then model the reference uncertainty of the *RId* using the machinery that we introduced earlier.

This assumes that our background theory gives us the true set of individuals and we are simply mapping the references to this set. An alternative description of the problem is that we have a set of references that we essentially need to partition into equivalence classes such that all the references in one equivalence class are referring to the same individual. In this case we don't explicitly map the reference to the individual.

We can model this by introducing a *SameRef* relation that uses some similarity function to determine when two references are the same:

$$SameRef(RefId1, RefId2) \leftarrow Author(RefId1, PId1), \\ Author(RefId2, PId2), Sim(RefId1, RefId2, \epsilon)$$

where *Sim* is a predicate which returns true when *RefId*1 and *RefId*2 are within $\epsilon$ of each other. The definition of the predicate is domain dependent. In its simplest form, it might be a string edit distance between the names. It might make use of domain knowledge about common names as well. For more complicated inferences, other modeling approaches are possible (Milch et al., 2004).

## 6. Comparison with other approaches

The two models that come closest to our approach are Bayesian Logic Programs (BLP) and Logical Bayesian Networks (LBN). We cannot describe these approaches in detail here so instead we go through an example, attempting to highlight both commonalities and differences.

We use a nice example from Fierens et al. (2004).

*There are students and courses. We know which students take which courses. Each student has an IQ and each course is either interesting or not. A student taking a certain course gets a grade for that course. Our belief about the interestingness of a course is influenced by the sum of the IQs of all students taking the course. Our belief about a student's grade for a course is influenced by the student's IQ.*

Note that by using the sum of the students' IQs, we do not differentiate between a larger class of students with low IQs and a smaller class with high IQs. Counting the number of students or taking the average IQs of the students may be more realistic, and these types of aggregates are easy to define in PRL.

We reproduce the structure of an example from Fierens et al. (2004) in Figure 3. We see for instance that Pete's grade in the course LP depends on Pete's IQ. And the interestingness of the course LP depends on Pete's IQ and Rick's IQ.

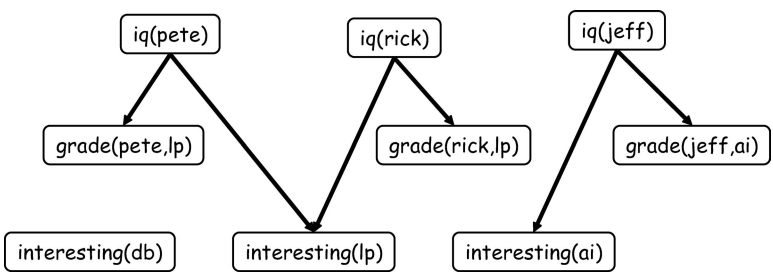


**Fig. 3.** The Bayesian network created for an LBN example from Fierens et al.

A BLP consists of a set of Bayesian clauses and background facts. For this example the Bayesian clauses are as follows:

$iq(S) \mid student(S)$

$interesting(C) \mid takes(S, C), iq(S)$

$interesting(C) \mid course(C)$

$grade(S, C) \mid iq(S), takes(S, C)$

The last clause, for instance, indicates the dependence of a student's grade in a course on the student's IQ. The ground atoms represent the random variables and are the ground atoms in the least Herbrand model of the BLP. The conditional dependencies of the Bayesian network are encoded by the ground instances of the BLP, so that, for instance, grade(pete,lp) depends on iq(pete) and takes(pete,lp). However, the ground atom takes(pete,lp) is not shown in the figure. Note also how the second clause shows the dependency of the interestingness of a class on the IQs of the students taking the course, but not how this is computed.

Fierens et al. (2004) note several disadvantages of BLPs. First, the BLP includes random variables that denote deterministic structural information in the network. For instance, student(jeff) is one of the background facts, yet it is also a random variable. Second, because BLPs combine probabilistic and deterministic (structural) information, it is hard to reconstruct the original description of the example.

In contrast with BLPs, LBNs use two types of predicates:logical and probabilistic. Logical predicates specify the deterministic information, while the probabilistic predicates represent the random variables. The ground atoms such as *takes*(*pete*,*lp*), using the predicates student, course, and takes, provide the background facts. In general, an arbitrary logic program is allowed for this purpose. A special type of clause is used to generate the random variables. In this example, this component consists of:

$iq(S) \Leftarrow student(S)$

$interesting(C) \Leftarrow course(C)$

$grade(S, C) \Leftarrow takes(S, C)$

For instance, the first clause states that there is a random variable, $iq(S)$, for each student $S$.

The probabilistic dependencies are represented by another component, which in this case would be:

$interesting(C) \mid iq(S) \leftarrow takes(S, C)$

$grade(S, C) \mid iq(S)$

The first clause is interpreted as stating that "whether a course is interesting depends on the IQ of a student, if the student takes that course".

The Bayesian network generated by the above LBN corresponds exactly to the Bayesian network shown in Figure 3. LBNs also define logical CPDs, which, like CPDs in PRL, are functions mapping a tuple from the parents of a node (representing a random variable) to a probability distribution on the node. For example, if the sum of the IQs of all students taking a certain course is greater than 1000, the probability that the course is interesting is 0.7, otherwise, it is 0.5.

Let us now consider how we would represent this example in a PRL program. We would have the following basic predicates:

*Student*(*Sname, Iq*)

*Course*(*Cname, Interesting*)

*Takes*(*Sname, Cname, Grade*)

PRL programs are closer in spirit to LBNs rather than BLPs. One difference is that PRLs make a somewhat stronger distinction between predicates such as *student*(*S*) and functions such as iq(*S*), although Fierens et al. (2005) distinguishes "probabilistic predicates" and "logical predicates" but admits that "probabilistic predicates are really functors". There are some syntactic differences also. For example, LBN uses three kinds of implication: $\Leftarrow$, $\leftarrow$, and ! while PRL uses only logical implication.

Some of the differences are that PRL introduces aggregate predicates explicitly. Also, PRL allows for the representation of various types of uncertainties in a natural way, as shown in the previous section. In the above example, we may want to say that high IQ students are more likely to enroll in difficult courses and difficult courses are more likely to be interesting. Then by observing only the registration lists for the courses and the grades for students, we can use probabilistic inference to reason about the IQ of the students: because the logical structure of the registration is actually part of the probabilistic model, the registrations give us evidence for the IQs of the students. Alternatively, given the IQs of the students and the difficulty of the courses, we can reason about the expected size of the enrollments and use these to make room assignments.

An interesting difference between BLPs and LBNs is that LBNs allow negated predicates in the body of a clause. Fierens et al. (2004) describe the use of negation to control the introduction of random variables. They give the following example

$$grade(S, C) \Leftarrow takes(S, C), not(absent(S, C))$$

which generates random variables for the grade of a student only if the student takes a class and is not absent. In some cases, the use of a predicate like absent may not be appropriate; after all, typically students who are absent also get grades in a class. Perhaps using *audit* instead of *takes* and not *absent* makes the example more intuitive. Students who take the course get a grade, while students who audit a class do not get a grade. Regardless, we would handle such an example in PRL by including a predicate *Audit*(*Sname*,*Cname*) for the students who are in the class but are auditing it instead of taking it for grade. Thus we solve the problem without the use of nonmonotonic negation, allowing us to keep Datalog semantics (of course the same could be done in LBNs; our point here is to avoid the nonmonotonic negation).

## 7. Conclusion

In this paper, we introduced PRL, a probabilistic relational language, which provides a powerful formalism for combining probability and logic. PRL is a reformulation of many of the ideas introduced in PRMs into a probabilistic logical language. We believe it helps to bridge the gap between frame-based approaches and probabilistic logic programming approaches and highlights many of the commonalities. Further, we have showed how it helps

to enable transfer of ideas between the two approaches, for example definitions of aggregates and structural uncertainty. We introduced an extended example, the research world domain, and showed how it could be framed naturally in a PRL program.

There are a few topics that we did not cover. We did not discuss class uncertainty (Getoor, 2001), but we believe that it also can be accommodated in a straightforward manner. Likewise, number uncertainty (Pfeffer, 2000) can also be modeled. The important topics of inference and learning are not covered here; the techniques developed in previous work is all applicable. And while we discussed approaches to structural uncertainty that do not make the unique names assumption, we are making a closed world assumption, unlike some recent proposed approaches (Milch et al., 2004; Laskey, 2003).

**Acknowledgments** We are grateful to the editors and referees for many very helpful comments and suggestions. This work was supported by NSF Grant 0308030 and supported and monitored by the Advanced Research and Development Activity (ARDA) and the National Geospatial-Intelligence Agency (NGA) under Contract Number CG0323.

## References

Bhattacharya, I. & Getoor, L. (2004). Iterative record linkage for cleaning and integration. In *9th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*.

Blockeel, H. (2003). Prolog for bayesian networks: A meta-interpreter approach. In *Proceedings of the 2nd Workshop on Multi-Relational Data Mining*.

Costa, V. S., Page, D. Qazi, & M. Cussens, J. (2003). CLP (BN): Constraint logic programming for probabilistic knowledge. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*. (pp. 517–524).

DeGroot, M. H. (1970)., *Optimal Statistical Decisions*. New York: McGraw-Hill.

Fierens, D., Blockeel, H., Bruynooghe, & M. Ramon, J. (2004). Logical bayesian networks. In *Proceedings of the 3rd Workshop on Multi-Relational Data Mining*.

Fierens, D., Blockeel, H., Bruynooghe, M. & Ramon, J. (2005). Logical bayesian networks and their relation to other probabilistic logical models. In *Proceedings of the Inductive Logic Programming Conference*.

Friedman, N., Getoor, L., Koller, D. & Pfeffer, A. (1999). Learning probabilistic relational models. In *Proceedings of the International Joint Conference on Artificial Intelligence*. Stockholm, Sweden (pp. 1300–1307) Morgan Kaufman.

Gaifman, H. (1964). Concerning measures in first order calculi. *Israel Journal of Mathematics, 2*, 1–18.

Getoor, L. (2001). Learning Statistical Models of Relational Data. Ph.D. thesis, Stanford University.

Getoor, L., Friedman, N., Koller, D. & Pfeffer, A. (2001a) Learning probabilistic relational models. In S. Dzeroski and N. Lavrac (eds.), *Relational Data Mining* Kluwer (pp. 307–335).

Getoor, L., Friedman. N., Koller, D. & Taskar, B. (2001b). Learning probabilistic relational models with structural uncertainty. In *Proceedings of the International Conference on Machine Learning*. Morgan Kaufman (pp. 170—177).

Getoor, L., Friedman, N., Koller, D. & Taskar, B. (2002). Learning probabilistic models with link structure. *Journal of Machine Learning Research*, *3*, 679–707.

Getoor, L., Rhee, J., Koller, D. & Small, P. (2004). Understanding tuberculosis epidemiology using probabilistic relational models. *Artificial Integllicence in Medicine*, *30* (233–256).

Getoor, L., Segal, E., Taskar, B., & Koller, D. (2001c). Probabilistic models of text and link structure for hypertext classification. In *IJCAI Workshop on Text Learning: Beyond Supervision.*

Getoor, L., Taskar, B., & Koller, D. (2001d). Using probabilistic models for selectivity estimation. In *Proceedings of ACM SIGMOD International Conference on Management of Data*. ACM Press (pp. 461–472).

Halpern, J. (1990). An analysis of first-order logics of probability. *Artificial Intelligence Journal, 46*, 311–350.

Heckerman, D., Meek, C. & Koller, D. (2004). Probabilistic models for relational data. Technical Report 30, Microsoft Research, Microsoft Corporation.

Jaeger, M. (1998). Reasoning about infinite random structures with relational Bayesian networks. In *Proceedings of the Knowledge Representations Conference* (pp. 570–581).

Kersting, K. & De Raedt, L. (2001a). Adaptive Bayesian logic programs. In *Proceedings of the Inductive Logic Programming Conference*.

Kersting, K. & De Raedt, L. (2001b). Bayesian logic programs. Technical Report 151, Institute for Computer Science, University of Freiburg, Germany.

Koller, D., McAllester, D. & Pfeffer, A. (1997). Effective Bayesian inference for stochastic programs. In *Proceedings of the National Conference on Artificial Intelligence* (pp. 740–747).
Koller, D. & Pfeffer, A. (1998). Probabilistic frame-based systems. In *Proceedings of American Artificial Intelligence Conference*.
Laskey, K. B. (2003). MEBN: A logic for open-world probabilistic reasoning. Technical report, Department of Systems Engineering and Operations Research, George Mason University.
Lloyd, J. W. (1989). *Foundations of Logic Programming*. Springer.
McCallum, A. & Wellner, B. (2003). Toward conditional models of identity uncertainty with application to proper noun coreference. In *IJCAI Workshop on Information Integration on the Web*.
Milch, B., Marthi, B. & Russell, S. (2004). BLOG: Relational modeling with unknown objects. In *ICML 2004 Workshop on Statistical Relational Learning and Its Connections to Other Fields*.
Muggleton, S. (1996). Stochastic logic programs. In *Advnaces in Inductive Logic Programming* (pp. 254–264).
Pasula, H., Marthi, B., Milch, B., Russell, S. & Shpitser, I. (2003). Identity uncertainty and citation matching. In *Advances in Neural Information Processing*.
Pasula, H. & Russell, S. (2001). Approximate inference for first-order probabilistic languages. In *Proceedings of the International Joint Conference on Artificial Intelligence* (pp. 741–748).
Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann.
Pfeffer, A. (2000). Probabilistic reasoning for complex systems. Ph.D. thesis, Stanford University.
Pfeffer, A., Koller, D., Milch, B. & Takusagawa, K. (1999). SPOOK: A system for probabilistic object-oriented knowledge representation. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*.
Poole, D. (1993). Probabilistic horn abduction and Bayesian networks. *Artificial Intelligence Journal*, *64* (1), 81–129.
Russell, S. (2001). Identity uncertainty. In *Proceedings of International Fuzzy Systems Association*.
Sato, T. & Kameya, Y. (1997) PRISM: A symbolic-statistical modeling language. In *Proceedings of the International Joint Conference on Artificial Intelligence* (pp. 1330–1335).
Wellman, M., Breese, J. & Goldman, R. (1992). From knowledge bases to decision models. *Knowledge Engineering Review*, *7* (1), 35–53.