Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence

An Algebraic Prolog for Reasoning about Possible Worlds

Angelika Kimmig and **Guy Van den Broeck** and **Luc De Raedt**
Department of Computer Science, Katholieke Universiteit Leuven,
Celestijnenlaan 200A - bus 2402, 3001 Heverlee, Belgium
{angelika.kimmig, guy.vandenbroeck, luc.deraedt}@cs.kuleuven.be

Abstract

We introduce aProbLog, a generalization of the probabilistic logic programming language ProbLog. An aProbLog program consists of a set of definite clauses and a set of algebraic facts; each such fact is labeled with an element of a semiring. A wide variety of labels is possible, ranging from probability values to reals (representing costs or utilities), polynomials, Boolean functions or data structures. The semiring is then used to calculate labels of possible worlds and of queries.

We formally define the semantics of aProbLog and study the aProbLog inference problem, which is concerned with computing the label of a query. Two conditions are introduced that allow one to simplify the inference problem, resulting in four different algorithms and settings. Representative basic problems for each of these four settings are: is there a possible world where a query is true (SAT), how many such possible worlds are there (#SAT), what is the probability of a query being true (PROB), and what is the most likely world where the query is true (MPE). We further illustrate these settings with a number of tasks requiring more complex semirings.

1 Introduction

There is significant interest in probabilistic approaches to logic programming and several probabilistic variants of Prolog have been developed, such as ICL (Poole 2000), Dyna (Eisner, Goldlust, and Smith 2005), PRISM (Sato and Kameya 2001) and ProbLog (De Raedt, Kimmig, and Toivonen 2007). Essentially, all these languages are based on definite clause logic (pure Prolog) extended with facts labeled with probability values. The meaning of such programs is typically derived from Sato's distribution semantics (Sato 1995), which assigns a probability to every literal. The probability of a Herbrand interpretation, also called possible world, is simply the product of the probabilities of the literals occurring in this world. The key concept is the success probability, which is the probability that a query succeeds in a randomly selected world. It is defined as the sum of the probabilities of the possible worlds in which the query is true. Several algorithms for inferring this probability and for learning the parameters of such logics have been developed over the past 15 years.

While the ability to reason about the probability of possible worlds and queries is central to artificial intelligence, applications exist where probabilities alone do not suffice and where one would like to associate other labels to possible worlds and queries, such as costs, weights, utilities, counts, gradients and even functions or data structures. The first types of labels could be used when making decisions, the last are often useful for inference and learning.

Analyzing the probabilistic Prologs from an algebraic point of view reveals that the probabilities associated to facts and queries are essentially elements of $\mathbb{R}_{\geq 0}$ and the operations needed to compute the success probability of a query are addition and multiplication, which means that one is operating in the semiring $(\mathbb{R}_{\geq 0}, +, \times, 0, 1)$.

This raises the question as to 1) whether it is possible to generalize these probabilistic Prologs to use labels from different semirings, and if so, 2) what AI problems can algebraic Prolog solve. This paper positively answers the first question and also identifies a wide range of problems that can be solved within such an algebraic Prolog.

To answer the first question we introduce a semantics for the algebraic Prolog, called aProbLog, which generalizes the probabilistic programming language ProbLog. An aProbLog program consists of a set of definite clauses and a set of algebraic facts. These are facts that are labeled with elements of a commutative semiring $\mathcal{R}$. The label of a possible world or Herbrand interpretation is then simply the product (in $\mathcal{R}$) of the labels of the algebraic literals it contains. The label of a query is the sum (in $\mathcal{R}$) of the labels of the possible worlds in which the query succeeds. We then study the inference problem that is concerned with the computation of the query labels and identify two properties (disjoint-sum and neutral-sum) that allow one to simplify the problem.

Even though other works such as Dyna (Eisner, Goldlust, and Smith 2005) and semiring-based constraint logic programming (Bistarelli and Rossi 2001) have labeled facts with elements of a semiring, aProbLog is – to the best of the authors' knowledge – the first extension of Prolog that can tackle the disjoint- and neutral-sum-problems; cf. Section 5 for a more detailed discussion.

To answer the second question, we show how aProbLog can be used to tackle a wide variety of tasks, including basic inference tasks in probabilistic logic programming. Other applications of aProbLog include shortest path problems, sensitivity analysis, computing the gradient of aProbLog parameters, or computing a binary decision diagram representing all possible worlds where a query can be proven.

The paper is structured as follows. We formally introduce our algebraic Prolog in Section 2 and discuss a variety of tasks that can be modelled in Section 3. The key ideas and challenges of inference and corresponding algorithms are introduced in Section 4. Before concluding, we discuss related work in Section 5. We shall assume some familiarity with the Prolog programming language, see for instance (Flach 1994) for an introduction.

2 aProbLog

For a set J of ground facts, we define the set of literals $\mathrm{L}(J)$ and the set of interpretations $\mathcal{I}(J)$ as follows:

$$\mathrm{L}(J) = J \cup \{\neg f \mid f \in J\} \tag{1}$$

$$\mathcal{I}(J) = \{S \mid S \subseteq \mathrm{L}(J) \wedge \forall l \in J:\ l \in S \leftrightarrow \neg l \notin S\} \tag{2}$$

An algebraic Prolog (aProbLog) program consists of

- a *commutative semiring* $(\mathcal{A}, \oplus, \otimes, e^{\oplus}, e^{\otimes})$[1]
- a finite set of ground *algebraic facts* $\mathrm{F} = \{f_1, \ldots, f_n\}$
- a finite set BK of *background knowledge clauses*
- a *labeling function* $\alpha : \mathrm{L}(\mathrm{F}) \rightarrow \mathcal{A}$

Background knowledge clauses are definite clauses, but their bodies may contain negative literals for algebraic facts. Their heads may not unify with any algebraic fact.

The idea of splitting a logic program in a set of facts and a set of clauses goes back to Sato's distribution semantics (Sato 1995), where it is used to define a probability distribution over interpretations of the entire program in terms of a distribution over the facts. This is possible because a truth value assignment to the facts in F uniquely determines the truth values of all other atoms defined in the background knowledge. In the simplest case, as realized in ProbLog, this basic distribution considers facts to be independent random variables and thus multiplies their individual probabilities. aProbLog uses the same basic idea, but generalizes from the semiring of probabilities to general commutative semirings. The distribution semantics is defined for countably infinite sets of facts. However, this assumes that the basic distribution can be constructed from a series of finite distributions, which does not always hold in the generalized setting. We therefore require the set of ground algebraic facts in aProbLog to be finite.[2]

In aProbLog, the label of a complete interpretation $I \in \mathcal{I}(\mathrm{F})$ is defined as the product of the labels of its literals

$$\mathbf{A}(I) = \bigotimes_{l \in I} \alpha(l) \tag{3}$$

and the label of a set of interpretations $S \subseteq \mathcal{I}(\mathrm{F})$ as the sum of the interpretation labels

$$\mathbf{A}(S) = \bigoplus_{I \in S} \bigotimes_{l \in I} \alpha(l) \tag{4}$$

A *query* q is a finite set of algebraic literals and atoms from the Herbrand base[3], $q \subseteq \mathrm{L}(\mathrm{F}) \cup HB(\mathrm{F} \cup \mathrm{BK})$. We denote the set of interpretations where the query is true by $\mathcal{I}(q)$,

$$\mathcal{I}(q) = \{I \mid I \in \mathcal{I}(\mathrm{F}) \wedge I \cup \mathrm{BK} \models q\} \tag{5}$$

The label of query q is defined as the label of $\mathcal{I}(q)$,

$$\mathbf{A}(q) = \mathbf{A}(\mathcal{I}(q)) = \bigoplus_{I \in \mathcal{I}(q)} \bigotimes_{l \in I} \alpha(l). \tag{6}$$

As both operators are commutative and associative, the label is independent of the order of both literals and interpretations. Calculating this label is the central inference task of aProbLog. Clearly, considering all possible interpretations to evaluate Equation (6) directly is infeasible for all but the tiniest programs; we will discuss an alternative approach in Section 4.

3 aProbLog Tasks

Given the semantics of aProbLog, the question is now which AI problems can be represented and solved by aProbLog. We will present a broad range of example tasks, with their respective logic programs, semirings and labeling functions.

Example 1. Consider the following aProbLog program, where we directly attach the positive label $\alpha(f)$ to a fact f and define labels of negative literals as $\alpha(\neg f) = 1 - \alpha(f)$.

```
calls(X) :- alarm, hears_alarm(X).
alarm :- burglary.
alarm :- earthquake.

0.7 :: hears_alarm(john).
0.7 :: hears_alarm(mary).
0.05 :: burglary.
0.01 :: earthquake.
```

This is a program where the labels are probabilities, and thus a simple ProbLog example. It models a variation of the famous alarm Bayesian network. In the abstract, we posed a number of basic questions one could ask about any query for this program. We now answer these questions for the query `calls(mary)`. The probability that the query succeeds in a randomly sampled world is $0.95 \cdot 0.01 \cdot 0.7 + 0.05 \cdot 0.99 \cdot 0.7 + 0.05 \cdot 0.01 \cdot 0.7 = 0.04165$. The most likely world where the query succeeds is $\{\mathtt{hears\_alarm}(\mathtt{john}), \mathtt{hears\_alarm}(\mathtt{mary}), \mathtt{burglary}\}$, with probability 0.001995. There are six worlds where the query succeeds, so the answer for SAT is yes as well.

The semiring structures used to answer these questions are given in Table 1. While the probabilistic logic programming system ProbLog focuses on the PROB problem, it is neither able to solve the MPE nor the #SAT problems.

[1] That is, *addition* $\oplus$ and *multiplication* $\otimes$ are associative and commutative binary operations over the set $\mathcal{A}$, $\otimes$ distributes over $\oplus$, $e^{\oplus} \in \mathcal{A}$ is the neutral element with respect to $\oplus$, $e^{\otimes} \in \mathcal{A}$ that of $\otimes$, and for all $a \in \mathcal{A}$, $e^{\oplus} \otimes a = a \otimes e^{\oplus} = e^{\oplus}$.

[2] In principle it is possible to allow non-ground algebraic facts to compactly represent a set of ground facts if one also associates finite domains to the different arguments of predicates or if the program does not contain functors.

[3] the set of ground atoms that can be constructed from the predicate, functor and constant symbols of the program

task	$\mathcal{A}$	$e^{\oplus}$	$e^{\otimes}$	$a \oplus b$	$a \otimes b$	$\alpha(f)$	$\alpha(\neg f)$
PROB	$\mathbb{R}_{\geq 0}$	0	1	$a+b$	$a \cdot b$	$\alpha(f_i) \in [0,1]$	$1-\alpha(f_i)$
MPE	$\mathbb{R}_{\geq 0}$	0	1	$\max(a,b)$	$a \cdot b$	$\alpha(f_i) \in [0,1]$	$1-\alpha(f_i)$
MPE State	$\mathbb{R}_{\geq 0} \times 2^{\{J \mid J \subseteq I \in \mathcal{I}(\mathrm{F})\}}$	$(0,\emptyset)$	$(1,\{\emptyset\})$	Eq. (8)	Eq. (7)	$\alpha(f_i) = (p_i, \{\{f_i\}\})$ with $p_i \in [0,1]$	$(1-p_i, \{\{\neg f_i\}\})$
SAT	$\{true, false\}$	$false$	$true$	$a \vee b$	$a \wedge b$	$\alpha(f_i) = true$	$true$
#SAT	$\mathbb{N}$	0	1	$a+b$	$a \cdot b$	$\alpha(f_i) = 1$	1
BDD	$\mathcal{BDD}(\mathcal{V})$	$bdd(0)$	$bdd(1)$	$a \vee_{bdd} b$	$a \wedge_{bdd} b$	$\alpha(f_i) = bdd(b_i)$	$\neg_{bdd} bdd(b_i)$
sensitivity	$\mathbb{R}[X]$	0	1	$a+b$	$a \cdot b$	$\alpha(f_i) = x$ or $\alpha(f_i) \in [0,1]$	$1-\alpha(f_i)$
gradient	$\mathbb{R}_{\geq 0} \times \mathbb{R}$	$(0,0)$	$(1,0)$	Eq. (9)	Eq. (10)	Eq. (11)	Eq. (12)

Table 1: Semiring definitions and labeling functions for the examples discussed in Section 3.

The MPE-State semiring in Table 1 extends the MPE semiring to return the set of corresponding worlds as a second argument. Its operators are defined as

$$(p,S) \otimes (q,T) = (p \cdot q, \{I \cup J \mid I \in S, J \in T\}) \quad (7)$$

$$(p,S) \oplus (q,R) = \begin{cases} (p,S) & \text{if } p > q \\ (q,R) & \text{if } p < q \\ (p, S \cup R) & \text{if } p = q \end{cases} \quad (8)$$

Note that multiplication is only well-defined if the resulting interpretations are consistent, which is guaranteed here as we only multiply labels of literals within possible worlds.

Further problems of interest for the alarm program include cases where the labels could be functions or even data structures. The algebraic definitions for the following examples are again given in Table 1.

We can for instance ask for a compact description of all worlds where the query is true. Boolean functions over a set of variables $\mathcal{V}$ can be represented as binary decision diagrams (BDDs, cf. Section 4.2 for more details). For a fixed variable order, there is a unique BDD for each such function. We can thus use the BDD semiring to describe sets of possible worlds, where $\mathcal{BDD}(\mathcal{V})$ is the set of BDDs over $\mathcal{V}$ for a fixed order, the function $bdd(\cdot)$ maps constants $true$ and $false$ and variables to their BDD representation, and $\vee_{bdd}$, $\wedge_{bdd}$, $\neg_{bdd}$ are the usual logical operators on BDDs.

Another task is sensitivity analysis, where probability labels are modeled by polynomials to investigate how changes in parameters influence the query probabilities.

Example 2. Replace the probability of `burglary` by x and that of `hears_alarm(mary)` by y. The probability of the query `calls(mary)` then becomes $0.99 \cdot x \cdot y + 0.01 \cdot y$.

To the best of our knowledge, sensitivity analysis for computing the success probabilities of queries is a novel task within probabilistic logic programming.

As a final example, the gradient of the success probability, as used for parameter learning in ProbLog (Gutmann et al. 2008), can directly be calculated in aProbLog as well, using the gradient semiring (Eisner 2002). We discuss the partial derivative with respect to the kth variable here, an extension to compute all partial derivatives in parallel is straightforward. The elements of this semiring are tuples, where the first element is the probability, the second its derivative. The binary operators are defined as follows:

$$(a_1, a_2) \oplus (b_1, b_2) = (a_1 + b_1, a_2 + b_2) \quad (9)$$

$$(a_1, a_2) \otimes (b_1, b_2) = (a_1 \cdot b_1, a_1 \cdot b_2 + a_2 \cdot b_1) \quad (10)$$

Multiplication uses the chain rule. The labeling functions are defined as follows, where $p_i \in [0,1]$ is the probability of f_i:

$$\alpha(f_i) = \begin{cases} (p_i, 1) & \text{if } i = k \\ (p_i, 0) & \text{if } i \neq k \end{cases} \quad (11)$$

$$\alpha(\neg f_i) = \begin{cases} (1-p_i, -1) & \text{if } i = k \\ (1-p_i, 0) & \text{if } i \neq k \end{cases} \quad (12)$$

While we have discussed tasks in probabilistic programming so far, aProbLog is not restricted to this setting.

Example 3. The following program together with the semiring $(\mathbb{N}, \min, +, \infty, 0)$ and $\alpha(\neg f) = 0$ for all algebraic facts calculates shortest paths based on travel times:

```
travel(X,Y) :- train(X,Y).
travel(X,Y) :- train(X,Z), travel(Z,Y).

135 :: train(london,paris).
82 :: train(paris,brussels).
113 :: train(brussels,amsterdam).
187 :: train(paris,cologne).
159 :: train(cologne,amsterdam).
107 :: train(brussels,cologne).
```

Using query `travel(london,amsterdam)`, the answer is $\min(135 + 82 + 113 + 0 + 0 + 0,\ 135 + 0 + 0 + 187 + 159 + 0, \ldots) = 330$.

When interpreting the labels as the capacity of the trains, we get the size of the biggest group that can travel together by using the semiring $(\mathbb{N}, \max, \min, -\infty, \infty)$ and $\alpha(\neg f) = \infty$. The same query now returns the maximum of among others $\min(135, 82, 113, \infty, \infty, \infty)$ and $\min(135, \infty, \infty, 187, 159, \infty)$, which is 135.

4 aProbLog Inference

The label $\mathbf{A}(q)$ of a query q is defined in terms of the set of possible worlds in which the query is true, that is, the worlds which allow for at least one derivation or proof of the query. The key to inference in aProbLog lies in using a compact description of this set when calculating labels. We will base this description on partial interpretations of the set of algebraic literals. The set of all possible (not necessary minimal) *explanations* of query q is defined as

$$\mathcal{X}(q) = \{R \mid R \subseteq I \in \mathcal{I}(\mathrm{F}) \wedge R \cup \mathrm{BK} \models q\} \quad (13)$$

A set $S \subseteq \mathcal{X}(q)$ is called a *covering explanation set* for query q if $\forall I \in \mathcal{I}(q)\ \exists J \in S : \ J \subseteq I$. The following discussion applies to an arbitrary covering explanation set $\mathcal{E}(q)$. In our algorithms, we construct $\mathcal{E}(q)$ based on standard Prolog inference using SLD resolution. Each SLD proof of query q results in an explanation containing all algebraic literals that are used in that proof, and $\mathcal{E}(q)$ is obtained by considering all proofs of q. The resulting set is a covering set of explanations, as each interpretation $I \in \mathcal{I}(q)$ allows for at least one proof.

Example 4. For the alarm program of Example 1 and the query `calls(mary)`, SLD resolution finds two proofs. This results in the explanation set $\{\{\mathtt{b}, \mathtt{h(m)}\}, \{\mathtt{e}, \mathtt{h(m)}\}\}$, where we abbreviate predicate and constant names using first letters. These two explanations cover all six interpretations in $\mathcal{I}(\mathtt{calls(mary)})$ and are therefore a covering explanation set.

Given $\mathcal{E}(q)$, the *explanation sum* $\mathbf{S}(\mathcal{E}(q))$ is defined as

$$\mathbf{S}(\mathcal{E}(q)) = \bigoplus_{E \in \mathcal{E}(q)} \bigotimes_{l \in E} \alpha(l) \quad (14)$$

Note that for $\mathcal{I}(q)$, which is a covering explanation set, the explanation sum coincides with $\mathbf{A}(q)$. In general, however, we note two differences in the definitions of these functions. First, the product in $\mathbf{S}(\mathcal{E}(q))$ ranges over subsets of algebraic facts only, thus covering multiple worlds, and second, the sum ranges over *sets* of possible worlds, which might overlap.

To address the first point, we define the label of an explanation $E \in \mathcal{X}(q)$ in analogy to the label of a query as

$$\mathbf{A}(E) = \mathbf{A}(\mathcal{I}(E)) = \bigoplus_{I \in \mathcal{I}(E)} \bigotimes_{l \in I} \alpha(l) \quad (15)$$

where $\mathcal{I}(E)$ is the set of interpretations where E is true:

$$\mathcal{I}(E) = \{I \mid I \in \mathcal{I}(\mathrm{F}) \wedge E \subseteq I\} \quad (16)$$

$\mathbf{A}(E)$ is called a *neutral sum* if

$$\mathbf{A}(E) = \bigotimes_{l \in E} \alpha(l), \quad (17)$$

that is, it can be calculated based on the literals in E only.

Example 5. Consider our earlier alarm example. The explanation $E = \{\mathtt{b}, \mathtt{e}, \mathtt{h(m)}\}$ is true in two interpretations and its label is calculated as

$$\begin{aligned}\mathbf{A}(E) &= \mathbf{A}(\{\mathtt{b}, \mathtt{e}, \mathtt{h(m)}, \mathtt{h(j)}\}) \oplus \mathbf{A}(\{\mathtt{b}, \mathtt{e}, \mathtt{h(m)}, \neg\mathtt{h(j)}\}) \\ &= (\alpha(\mathtt{b}) \otimes \alpha(\mathtt{e}) \otimes \alpha(\mathtt{h(m)}) \otimes \alpha(\mathtt{h(j)})) \\ &\oplus (\alpha(\mathtt{b}) \otimes \alpha(\mathtt{e}) \otimes \alpha(\mathtt{h(m)}) \otimes \alpha(\neg\mathtt{h(j)})) \\ &= (0.00035 \cdot 0.7) \oplus (0.00035 \cdot 0.3)\end{aligned}$$

In the PROB semiring, $\alpha(\mathtt{h(j)}) \oplus \alpha(\neg\mathtt{h(j)}) = 0.7 + 0.3 = 1$, and thus $\mathbf{A}(E) = \alpha(\mathtt{b}) \otimes \alpha(\mathtt{e}) \otimes \alpha(\mathtt{h(m)})$. This is not true in the MPE semiring, where $\alpha(\mathtt{h(j)}) \oplus \alpha(\neg\mathtt{h(j)}) = \max(0.7, 0.3) = 0.7 \neq 1$.

	neutral sums	NSP
disjoint sums	SAT	MPE
DSP	PROB	#SAT

Table 2: Classification of four example tasks: if sums are not disjoint, the disjoint-sum-problem (DSP) occurs; if they are not neutral, the neutral-sum-problem (NSP) occurs.

Property 1. *If* $\forall f \in \mathrm{F} : \ \alpha(f) \oplus \alpha(\neg f) = e^{\otimes}$*, then the sum* $\mathbf{A}(E)$ *is neutral.*

To address the second observation above, we now consider the sum in (14). The sum $\bigoplus_{E \in \mathcal{E}(q)} \mathbf{A}(E)$ is called a *disjoint sum* if

$$\bigoplus_{E \in \mathcal{E}(q)} \mathbf{A}(E) = \bigoplus_{I \in \mathcal{I}(q)} \mathbf{A}(I). \quad (18)$$

Example 6. Consider again the alarm example, ignoring the $\mathtt{h(X)}$ facts for the sake of brevity. The set of explanations for query `alarm` is $\{\{\mathtt{e}\}, \{\mathtt{b}\}\}$, and the set of interpretations where the query is true is $\{\{\mathtt{e}, \mathtt{b}\}, \{\mathtt{e}, \neg\mathtt{b}\}, \{\neg\mathtt{e}, \mathtt{b}\}\}$. We get $\mathbf{A}(\{\mathtt{e}\}) + \mathbf{A}(\{\mathtt{b}\}) = (0.05 \cdot 0.01 + 0.05 \cdot 0.99) + (0.01 \cdot 0.05 + 0.01 \cdot 0.95) = 0.06$, while the sum over interpretations is $0.95 \cdot 0.01 + 0.05 \cdot 0.99 + 0.05 \cdot 0.01 = 0.0595$ only. The sum is thus not disjoint. However, if we use the MPE semiring instead, the sum is disjoint, as maximization is not affected by repeatedly summing the label of the same interpretation.

Property 2. *If* $\oplus$ *is idempotent*[4]*, then* $\bigoplus_{E \in \mathcal{E}(q)} \mathbf{A}(E)$ *is a disjoint sum.*

Note that this is a sufficient condition only. Equation (18) also holds in other cases, for instance, if explanations in $\mathcal{E}(q)$ are mutually exclusive, that is, at most one of them exists in any possible world.

Property 3. *If* $\mathbf{A}(E)$ *is a neutral sum for all* $E \in \mathcal{E}(q)$ *and* $\bigoplus_{E \in \mathcal{E}(q)} \mathbf{A}(E)$ *is a disjoint sum, the explanation sum equals the query label, that is,* $\mathbf{S}(\mathcal{E}(q)) = \mathbf{A}(q)$.

In this case, inference can directly evaluate $\mathbf{S}(\mathcal{E}(q))$ based on (14), which is straightforward. Otherwise, it will be necessary to address the *neutral-sum-problem* and/or the *disjoint-sum-problem* during inference. These are the key dimensions along which inference settings in aProbLog are characterized. The four inference tasks of Example 1 are characteristic for the resulting four settings, cf. Table 2. We will now discuss the two problems and introduce algorithms for all four inference settings.

4.1 Neutral-sum-problem

We denote the set of variables not occurring in an explanation E by

$$\mathrm{free}(E) = \{f \mid f \in \mathrm{F} \wedge f \not\in E \wedge \neg f \not\in E\}. \quad (19)$$

Using this set, we obtain

$$\mathbf{A}(E) = \bigotimes_{l \in E} \alpha(l) \otimes \bigotimes_{l \in \mathrm{free}(E)} (\alpha(l) \oplus \alpha(\neg l)) \quad (20)$$

[4] $\oplus$ is idempotent if $a \oplus a = a$ for all $a \in \mathcal{A}$

as a direct consequence of the definition of $\mathbf{A}(E)$ and the properties of commutative semirings.

The following property forms the basis of the algorithm addressing the neutral-sum-problem.

Property 4. *Let* $\mathrm{V_i} = \{f \mid f \in E_i \vee \neg f \in E_i\}$, *then*

$$\mathbf{A}(E_0) \oplus \mathbf{A}(E_1) = (\mathbf{P}_1(E_0) \oplus \mathbf{P}_0(E_1)) \otimes \bigotimes_{f \in \mathrm{F} \setminus (\mathrm{V}_0 \cup \mathrm{V}_1)} (\alpha(f) \oplus \alpha(\neg f))$$

where

$$\mathbf{P}_j(E_i) = \bigotimes_{l \in E_i} \alpha(l) \otimes \bigotimes_{f \in \mathrm{V_j} \setminus \mathrm{V_i}} (\alpha(f) \oplus \alpha(\neg f)) .$$

That is, to solve the neutral-sum-problem while evaluating $\bigoplus_{E \in \mathcal{E}(q)} \bigotimes_{l \in E} \alpha(l)$, it suffices to keep track of the set of variables the intermediate results are based on, and to take these into account before summing.[5] Labels of variables that do not occur in any $E \in \mathcal{E}(q)$ need only to be taken into account at the very end.

Example 7. Let us calculate the label of $\mathtt{calls(mary)}$ using the MPE semiring. The set of explanations obtained from resolution contains $\{\mathtt{b}, \mathtt{h(m)}\}$ and $\{\mathtt{e}, \mathtt{h(m)}\}$, so $\mathbf{S}(\mathcal{E}(q)) = \alpha(\mathtt{b}) \otimes \alpha(\mathtt{h(m)}) \oplus \alpha(\mathtt{e}) \otimes \alpha(\mathtt{h(m)})$. We obtain intermediate results $\alpha(\mathtt{b}) \otimes \alpha(\mathtt{h(m)}) = 0.05 \cdot 0.7 = 0.035$ and $\alpha(\mathtt{e}) \otimes \alpha(\mathtt{h(m)}) = 0.01 \cdot 0.7 = 0.007$, which we now want to sum. As they use different variables, we correct to $\mathbf{P}(\{\mathtt{b}, \mathtt{h(m)}\}) = 0.035 \times \max(0.01, 0.99) = 0.03465$, taking into account $\mathtt{e}$, and $\mathbf{P}(\{\mathtt{e}, \mathtt{h(m)}\}) = 0.007 \cdot \max(0.05, 0.95) = 0.00665$, taking into account $\mathtt{b}$. The sum of these two now is $\max(0.03465, 0.00665) = 0.03465$, corresponding to $\{\mathtt{b}, \mathtt{h(m)}, \neg\mathtt{e}\}$. As these are the only explanations, we finish by taking into account $\mathtt{h(j)}$, resulting in $0.03465 \cdot \max(0.7, 0.3) = 0.001995$ corresponding to $\{\mathtt{b}, \mathtt{h(m)}, \neg\mathtt{e}, \mathtt{h(j)}\}$.

4.2 Disjoint-sum-problem

In situations that require solving the disjoint-sum-problem, such as calculating success probabilities or gradients (bottom of Table 2), we follow ProbLog's approach of using Binary decision diagrams (BDDs) (Bryant 1986). A BDD is an efficient graphical representation of a Boolean function over a set of variables, and thus can be used to encode the set $\mathcal{E}(q)$, which maps truth value assignments for algebraic literals to the truth value of the query q. Given a fixed variable ordering, a Boolean function φ can be represented as a full Boolean decision tree where each node on the ith level is labeled with the ith variable and has two children called low and high. Each path from the root to some leaf stands for one complete variable assignment. If variable x is assigned 0 (1), the branch to the low (high) child is taken. Each leaf is labeled by the outcome of φ given the variable assignment represented by the corresponding path. Starting from such a tree, one obtains a BDD by merging isomorphic subgraphs and deleting redundant nodes until no further reduction is possible. A node is redundant iff the subgraphs rooted at its children are isomorphic.

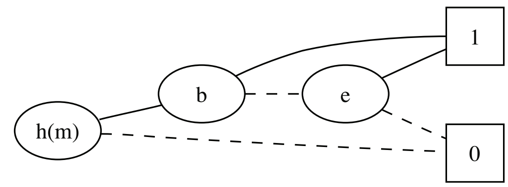

Figure 1: BDD for query `calls(mary)`.

Example 8. The BDD in Figure 1 encodes $\mathcal{E}(\mathtt{calls(mary)}) = \{\{\mathtt{b}, \mathtt{h(m)}\}, \{\mathtt{e}, \mathtt{h(m)}\}\}$. Dashed edges indicate 0's and lead to low children. Solid ones indicate 1's and lead to high children.

Given such a BDD, the query label can be calculated as the label of the BDD's root node according to the following recursive definition, where h and l denote the high and low child of node n, respectively:

$$\mathbf{label}(1) = e^{\otimes}$$
$$\mathbf{label}(0) = e^{\oplus}$$
$$\mathbf{label}(n) = (\alpha(n) \otimes \mathbf{label}(h)) \oplus (\alpha(\neg n) \otimes \mathbf{label}(l))$$

Example 9. When calculating probabilities in Figure 1, we get $\mathbf{label}(\mathtt{e}) = 0.01 \cdot 1 + 0.99 \cdot 0 = 0.01$, $\mathbf{label}(\mathtt{b}) = 0.05 \cdot 1 + 0.95 \cdot 0.01 = 0.0595$, and $\mathbf{label}(\mathtt{h(m)}) = 0.7 \cdot 0.0595 + 0.3 \cdot 0 = 0.04165$.

If intermediate results are cached, the algorithm has a time and space complexity linear in the size of the BDD. Calculating the label on the BDD exploits associativity and commutativity of both semiring operations (as variables need to be brought in the same order on all paths through the BDD, and paths are sorted according to variable values) as well as distributivity (to obtain a tree-shaped representation of the formula). Merging isomorphic subtrees does not affect the calculation, as all ingoing edges are maintained. If sums are neutral, dropping redundant nodes does not affect the result, as in this case the omitted update $(\alpha(n) \otimes \mathbf{label}(s)) \oplus (\alpha(\neg n) \otimes \mathbf{label}(s))$ equals $\mathbf{label}(s)$. If these sums are not neutral, we again use Property 4 to modify the inference algorithm. This results in Algorithm 1, which addresses both the neutral- and the disjoint-sum-problem (bottom right of Table 2). The algorithm returns both the current label and the underlying set of algebraic facts. Lines 9 and 10 perform the updates for the labels of both subtrees based on Property 4, restricted to those variables that appear in one of the subtrees. The updates with respect to variables that do not appear in the BDD are not included in the algorithm. They have to be performed in a post-processing step.

Note that this general algorithm is correct in all four inference settings, but simpler versions as discussed before can be used if sums are neutral or disjoint. If sums are neutral but not disjoint, Algorithm 1 can be simplified by 1) restricting return values to their first argument and 2) using the labels of the children directly in line 11, thus dropping lines 9

[5] This idea is closely related to the concept of a smooth NNF circuit used in knowledge compilation, where operands of a disjunction use the same set of variables (Darwiche and Marquis 2002).

Algorithm 1 General aProbLog inference algorithm.

1: **function** $\text{LABEL}(\text{BDD node } n)$
2: **if** n is the 1-terminal **then**
3: return $(e^{\otimes}, \emptyset)$
4: **if** n is the 0-terminal **then**
5: return $(e^{\oplus}, \emptyset)$
6: let h and l be the high and low children of n
7: $(H, \mathrm{V_h}) := \text{LABEL}(h)$
8: $(L, \mathrm{V_l}) := \text{LABEL}(l)$
9: $\mathbf{P}_l(h) := H \otimes \bigotimes_{x \in \mathrm{V_l} \setminus \mathrm{V_h}} (\alpha(x) \oplus \alpha(\neg x))$
10: $\mathbf{P}_h(l) := L \otimes \bigotimes_{x \in \mathrm{V_h} \setminus \mathrm{V_l}} (\alpha(x) \oplus \alpha(\neg x))$
11: $\mathbf{label}(n) := (\alpha(n) \otimes \mathbf{P}_l(h)) \oplus (\alpha(\neg n) \otimes \mathbf{P}_h(l))$
12: return $(\mathbf{label}(n), \{n\} \cup \mathrm{V_h} \cup \mathrm{V_l})$

and 10. If the sum is disjoint, it is not necessary to build a BDD, but one can instead iterate over the set $\mathcal{E}(q)$ directly as discussed above, using Property 4 if sums are not neutral.

We have implemented all four algorithms based on the publicly available implementation of ProbLog (Kimmig et al. 2011) and have tested them with the semirings discussed here.

5 Related Work

The weighted logic programming language Dyna (Eisner, Goldlust, and Smith 2005), semiring-based constraint logic programming (Bistarelli and Rossi 2001) and weighted Datalog (Bistarelli, Martinelli, and Santini 2008) all extend (a variation) of a logic programming language by associating labels from a semiring to facts. However, aProbLog is the first such extension of Prolog that addresses both the neutral- and the disjoint-sum-problem. In contrast to these other systems, aProbLog bases query labels on interpretations, not proofs. This allows us to directly cover the PROB and MPE tasks that require to consider possible worlds. On the other hand, problems such as shortest path can be expressed in both semantics. Semiring-based constrained logic programming and weighted Datalog require semiring addition to be idempotent. Thus, the disjoint-sum-problem does not arise (cf. Property 2). While Dyna does not impose such restrictions, it does not address the disjoint-sum-problem either. This rules out for instance the semirings for PROB (if possible worlds can contain multiple proofs), #SAT, sensitivity analysis and gradient and thus even the ProbLog setting. As the other systems do not consider full possible worlds, they also avoid the neutral-sum-problem, which is concerned with basic facts not occurring in a proof.

6 Conclusions

We have introduced aProbLog, an algebraic Prolog where facts are labeled with elements of a semiring. Labels of possible worlds and labels of queries are then defined within this semiring. We have identified two conditions that allow for simplification of the inference task in aProbLog, that is, the task of calculating query labels. This led to four different settings, for which we have also introduced corresponding algorithms. Furthermore, we have discussed a broad range of tasks that can be modeled and solved in aProbLog, thereby illustrating the potential of this new framework.

Acknowledgements. Angelika Kimmig and Guy Van den Broeck are supported by the Research Foundation-Flanders (FWO-Vlaanderen). This work is partially supported by the GOA/08/008 Probabilistic Logic Learning and by the European Commission under the 7th Framework Program, contract no. BISON-211898.

References

Bistarelli, S., and Rossi, F. 2001. Semiring-based constraint logic programming: syntax and semantics. *ACM Trans. Program. Lang. Syst.* 23:1–29.

Bistarelli, S.; Martinelli, F.; and Santini, F. 2008. Weighted Datalog and levels of trust. In *Int. Conference on Availability, Reliability and Security*, 1128 –1134.

Bryant, R. E. 1986. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers* 35(8):677–691.

Darwiche, A., and Marquis, P. 2002. A knowledge compilation map. *Journal of Artificial Intelligence Research* 17(1):229–264.

De Raedt, L.; Kimmig, A.; and Toivonen, H. 2007. ProbLog: A probabilistic Prolog and its application in link discovery. In Veloso, M. M., ed., *Proc. of the Int. Joint Conference on Artificial Intelligence*, 2462–2467.

Eisner, J.; Goldlust, E.; and Smith, N. 2005. Compiling Comp Ling: Weighted dynamic programming and the Dyna language. In *Proc. of the Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, 281–290. ACL.

Eisner, J. 2002. Parameter estimation for probabilistic finite-state transducers. In *Proc. of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, 1–8.

Flach, P. 1994. *Simply logical - Intelligent Reasoning by Example*. John Wiley.

Gutmann, B.; Kimmig, A.; Kersting, K.; and De Raedt, L. 2008. Parameter learning in probabilistic databases: A least squares approach. In Daelemans, W.; Goethals, B.; and Morik, K., eds., *Proc. of the European Conference on Machine Learning*, volume 5211 of *LNCS*, 473–488. Springer.

Kimmig, A.; Demoen, B.; De Raedt, L.; Santos Costa, V.; and Rocha, R. 2011. On the implementation of the probabilistic logic programming language ProbLog. *Theory and Practice of Logic Programming (TPLP)* 11:235–262.

Poole, D. 2000. Abducing through negation as failure: stable models within the independent choice logic. *Journal of Logic Programming* 44(1-3):5–35.

Sato, T., and Kameya, Y. 2001. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research (JAIR)* 15:391–454.

Sato, T. 1995. A statistical learning method for logic programs with distribution semantics. In Sterling, L., ed., *Proc. of the Int. Conference on Logic Programming*, 715–729. MIT Press.