# Tabling for P-log Probabilistic Query Evaluation

Han The Anh, Carroline Kencana Ramli, and Carlos Viegas Damásio
$\{h.anh|c.kencana\}@fct.unl.pt$, $cd@di.fct.unl.pt$

CENTRIA, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal.

**Abstract.** We propose a new approach for implementing P-log using XASP, the interface of XSB with Smodels. By using the tabling mechanism of XSB, our system is most of the times faster than P-log. In addition, our implementation has query features not supported by P-log, as well as new set operations for domain definition.

## 1 Introduction

Probabilistic reasoning became the major approach for reasoning under uncertainty. From the standpoint of logic and logic programming, the addition of probabilities allows us to represent and reason about finitely varying degrees of belief. From the standpoint of probability and Bayesian Networks, the addition of rule-based representations allows the creation and modification of probabilistic models more easily. The combination of these two lines of research has been attempted in the recent years, resulting in formalisms with both logical and probabilistic knowledge representation capabilities.

One of the approaches is Pearl's probabilistic causal models [8]. Many of the difficulties in probabilistic reasoning on Pearl's formalism lie not only in the use of probabilistic models, but in their formulation relying on propositional reasoning. Other formalisms have proposed the integration between logic and probability such as Pooles Probabilistic Horn Abduction (PHA) [3], the Independent Choice Logic (ICL) [4, 5], the LPAD formalism [9], and the recent language P-log [1].

P-log is a declarative language based on a logic formalism for probabilistic reasoning and action. P-log uses Answer Set Programming (ASP) as its logical and Causal Bayesian Networks (CBN) as its probabilistic foundations. A P-log program consists of a logical part and a probabilistic part. The logical part represents knowledge which determines the possible worlds of the program, including ASP rules and declarations of random attributes, while the probabilistic part contains probability declarations (pr-atoms) which determine the probabilities of those worlds [1, 2]. Although ASP has been proven to be a useful paradigm for solving a variety of combinatorial problems, its non-relevance property [11] makes the P-log system sometimes computationally redundant.

In this paper, we explore a new approach for implementing P-log using XASP, the interface of XSB with Smodels [11] – an answer set solver. With XASP, the relevance of the system is maintained [7]. Moreover, by using the tabling mechanism of XSB our system is most of the times faster than P-log. In addition, our implementation has new kind of queries not supported by P-log, as well as new set operations for domain definition.

The rest of this paper is organized as follows. Section 2 provides a description of the syntax and semantics of the extended P-log system, following closely the formalisms in [1, 2]. Section 3 exhibits some examples. Section 4 outlines the implementation, providing benchmarks comparing P-log(ASP) (original P-log implementation based on [1, 2]) and P-log(XSB) (the new implementation). The paper finishes with conclusions and directions for future work.

## 2 Extended P-log

In this section, the syntax and semantics of extended P-log programs are defined, being compatible with those ones of the original P-log [1]. The extended syntax has constructs for declaring new sorts by union or intersection of other sorts. This syntactic sugar enables a more declarative representation of many practical problems; for instance, the domain of students who attend both physics and math courses is the intersection of participants in each course, or the domain of cards in the poker game is the union of hearts, spades, diamonds and clubs. In addition, by using XASP, the logical part can use arbitrary XSB prolog code, thus, allowing for the representation of more complex problems that are more difficult or even impossible to express in the original P-log language. The semantics is given by an adapted program transformation into XASP, including the new set operations.

### 2.1 Syntax

In general, a P-log program $\Pi$ consists of a sorted signature, declarations, a regular part, a set of random selection rules, a probabilistic information part, and a set of observations and actions.

**Sorted signature and Declaration** The sorted signature $\Sigma$ of $\Pi$ contains a set of constant symbols and term-building function symbols, which are used to form terms in the usual way. Additionally, the signature contains a collection of special function symbols called attributes. Attribute terms are expressions of the form $a(\bar{t})$, where $a$ is an attribute and $\bar{t}$ is a vector of terms of the sorts required by $a$. A literal is an atomic statement, $p$, or its explicit negation, $neg\_p$. In addition, $p$ is considered to be the explicit negation of $neg\_p$. The expressions $p$ and $not\ p$ where $not$ is the default negation of ASP are called extended literals.

The declaration part of a P-log program is defined as a collection of sorts and sort declarations of attributes. A sort $c$ can be defined by listing all the elements $c = \{x_1, ..., x_n\}$, by specifying the range of values $c = \{L..U\}$ where $L$ and $U$ are the integer lower bound and upper bound, or even by specifying range of values of members $c = \{h(L..U)\}$ where $h/1$ is a unary predicate. We are also able to define a sort by arbitrarily mixing the previous constructions, e.g. $c = \{x_1, .., x_n, L..U, h(M..N)\}$. In addition, in the extended version, it is allowed to declare union and intersection of sorts: $c = union(c_1, ...., c_n)$ and $c = intersection(c_1, ..., c_n)$, respectively, where $c_i, 1 \leq i \leq n$, are declared sorts.

Declaration of an attribute $a$ with domain $c_1 \times ... \times c_n$ and range $c_0$ is represented by: $a : c_1 \times ... \times c_n$ --> $c_0$. If $a$ has no domain parameter, we simply write $a : c_0$. The range of $a$ is denoted by $range(a)$.

**Regular part** This part of a P-log program consists of a collection of XSB Prolog rules, facts and integrity constraints (IC), formed by literals of $\Sigma$. An IC is encoded as a XSB rule with the `false` literal in the head.

**Random Selection Rule** This is a rule for attribute $a$ having the form:

$$random(RandomName, a(\bar{t}), DynamicRange) \texttt{ :- } Body$$

This means the attribute instance $a(\bar{t})$ is random if the preconditions in $Body$ are satisfied. The $DynamicRange$ allows to restrict the default range for random attributes. The $RandomName$ is a syntactic mechanism used to link random attributes to the corresponding probabilities. If there is no precondition, we simply put $true$ in the body. A constant $full$ can be used in $DynamicRange$ to signal that the dynamic domain equals to $range(a)$.

**Probabilistic Information** Information about probabilities of random attribute instances $a(\bar{t})$ taking particular value $y$ is given by probability atoms (or pa-atoms for short) which have the following form:

$$pa(RandomName, a(\bar{t}, y), d(A, B)) \texttt{ :- } Body.$$

It means if $Body$ were true, and the value of $a(\bar{t})$ were selected by a rule named $RandomName$, then $Body$ would cause $a(\bar{t}) = y$ with probability $\frac{A}{B}$.

*Example 1 (Dice [1]).* There are two dice, `d1` and `d2`, belonging to Mike and John, respectively. Each dice has scores from 1 through 6, and will be rolled once. The dice owned by Mike is biased to 6 with probability $1/4$. This scenario can be coded with the following P-log(XSB) program $\Pi_{dice}$

```
1. score = {1..6}.   dice = {d1,d2}.
2. owns(d1,mike). owns(d2,john).
3. roll : dice --> score.
4. random(r(D), roll(D), full) :- true.
5. pa(r(D), roll(D,6), d(1,4)) :- owns(D,mike).
```

Two sorts $score$ and $roll$ of the signature of $\Pi_{dice}$ are declared in lines 1. The regular part contains two facts in line 2 saying that dice $d1$ belongs to Mike and $d2$ belongs to John. Line 3 is the declaration of attribute $roll$ mapping each dice to a score. Line 4 states that the distribution of attribute $roll$ is random. Line 5 belongs to probabilistic information part, saying that the dice owned by Mike is biased to 6 with probability $\frac{1}{4}$.

Note that the probability of an atom $a(\bar{t}, y)$ will be directly assigned if the corresponding `pa/3` atom is in the head of some *pa-rule* with a true body. To define probabilities of the remaining atoms we assume that by default, all values of a given attribute which are not assigned a probability are equally likely. For instance, probabilities of rolling Mike's dice to be $i, i \in \{1, 2, 3, 4, 5\}$, are the same and equal to 3/20.

**Observations and Actions** Observations and actions are, respectively, statements of the forms $obs(l)$ and $do(l)$, where $l$ is a literal. Observations are used to record the outcomes of random events, i.e. random attributes and attributes dependent on them. The dice domain may, for instance, contain `obs(roll(d1,4))` to record the outcome of rolling dice $d1$. The statement $do(a(t, y))$ indicates that $a(t) = y$ is made true as the result of a deliberate (non-random) action. For instance, `do(roll(d1,4))` may indicate that $d1$ was simply put on the table in the described position.

### 2.2 Semantics

The semantics is defined in two stages. First it defines a mapping of the logical part of $\Pi$ into its XASP counterpart $\tau(\Pi)$. The answer sets of $\tau(\Pi)$ plays the role of possible worlds of $\Pi$. Next the probabilistic part of $\tau(\Pi)$ is used to define a measure over the possible worlds as well as the probability of (complex) formulas. This part of the semantics exactly follows the one in [1].

The logical part of a P-log program $\Pi$ is transformed into its XASP counterpart $\tau(\Pi)$ by the following five steps:

1. **Sort declaration:**
   - for every sort declaration $c = \{x_1, .., x_n\}$ of $\Pi$, $\tau(\Pi)$ contains $c(x_i)$ for each $1 \leq i \leq n$.
   - for every sort declaration $c = \{L..U\}$ of $\Pi$, $\tau(\Pi)$ contains $c(i)$ where $L \leq i \leq U$, with integers $L \leq U$.
   - for every sort declaration $c = \{h(L..U)\}$ of $\Pi$, $\tau(\Pi)$ contains $c(h(i))$ where $L \leq i \leq U$, with integers $L \leq U$.
   - for every sort declaration $c = union(c_1, ..., c_n)$, $\tau(\Pi)$ contains the rules $c(X) :- c_i(X)$ for each $1 \leq i \leq n$.
   - for every sort declaration $c = intersection(c_1, ..., c_n)$, $\tau(\Pi)$ contains the rule $c(X) :- c_1(X), \ldots, c_n(X)$.
2. **Regular part:** For each attribute term $a(\bar{t})$, $\tau(\Pi)$ contains the rules:
   - $false :- a(\bar{t}, Y1),\ a(\bar{t}, Y2),\ Y1 \backslash = Y2.$
     which is to guarantee that in each answer set $a(\bar{t})$ has at most one value.
   - $a(\bar{t}, y) :- do(a(\bar{t}, y))$.
     which is to guarantee that the atoms which are made true by a deliberate action are indeed true.
3. **Random selection:**
   - For attribute $a$, $\tau(\Pi)$ contains the rule: $intervene(a(\bar{t})) :- do(a(\bar{t}, Y))$.
   - Each random selection rule
     $random(RanName, a(\bar{t}), DynRange) :- Body.$
     is translated into:
     - $a(\bar{t}, Y) :- tnot(intervene(a(\bar{t}))),\ tnot(neg\_a(\bar{t}, Y)),\ Body.$
     - $neg\_a(\bar{t}, Y) :- tnot(intervene(a(\bar{t}))),\ tnot(a(\bar{t}, Y)),\ Body.$
     - $atLeastOne(RanName, \bar{t}) :- a(\bar{t}, Y).$
     - $false :- tnot(atLeastOne(RanName, \bar{t}))$.
     - if $DynRange$ is $full$, $\tau(\Pi)$ contains
       $pd(RanName,\ a(\bar{t}, Y)) :- tnot(intervene(a(\bar{t}))),\ Body.$
     - if $DynRange$ is not $full$, $\tau(\Pi)$ contains two rules
       $false :- a(\bar{t}, Y),\ tnot(DynRange),\ Body,\ tnot(intervene(a(\bar{t})))$.
       $pd(RanName,\ a(\bar{t}, Y)) :- tnot(intervene(a(\bar{t}))),\ DynRange,\ Body.$
4. **Observation and action:** $\tau(\Pi)$ contains actions and observations of $\Pi$.
5. For each literal $l$, $\tau(\Pi)$ contains the rule: $false :- obs(l),\ tnot(l)$.

Similarly to ASP, in the body of each XASP rule additional domain predicates are necessary for grounding variables appeared in non-domain predicates (see example 2 for concrete details). In the transformation the XSB default table negation operator

$tnot$/1 is used. In the transformation of random selection the auxiliary predicate `pd/2` is used to define default probabilities, recording for each world what are the possible values $Y$ for random attribute term $a(\bar{t})$ . The execution of a deliberate action on $a(\bar{t})$ makes the corresponding $intervene(a(\bar{t}))$ true, thereby blocking the generation of random alternatives for attribute $a(\bar{t})$. Also notice that our semantics is equivalent to the semantics defined in [1] for the original P-log syntax. In fact, we reformulated the transformation from the original paper to adapt it to the XASP syntax. For example, the cardinality expression of Smodels language used in the original paper is replaced with an even loop to generate stable models and rules for determining upper and lower bounds. The rationale for the transformation can be found in [1].

Notice that the probabilistic information part of a P-log program, consisting of pa-rules, is kept unchanged through the transformation

*Example 2.* For better understanding of the transformation, we provide here the resulting transformed program $\tau(\Pi_{dice})$ of $\Pi_{dice}$ described in example 1:

```
1. score(1). score(2). score(3). score(4). score(5). score(6).
2. dice(d1). dice(d2). owns(d1,mike). owns(d2,john).
3. false:-score(X),score(Y),dice(D),roll(D,X),roll(D,Y),X \= Y.
4. roll(D,X) :- dice(D), score(X), do(roll(D,X)).
5. intervene(roll(D)):-dice(D),score(X),do(roll(D,X)).
6. roll(D,X) :- dice(D), score(X),
       tnot(intervene(roll(D))),tnot(neg_roll(D,X)).
7. neg_roll(D,X) :- dice(D), score(X),
       tnot(intervene(roll(D))),tnot(roll(D,X)).
8. atLeastOne(r(D),D) :- dice(D), score(X), roll(D,X).
9. false :- dice(D), tnot(atLeastOne(r(D),D)).
10.pd(r(D),roll(D,X)) :- dice(D), score(X),
       tnot(intervene(roll(D,X))).
11.pa(r(D),roll(D,6),d(1,4)) :- owns(D,mike).
```

Lines 1-2 are the transformation of sorts declaration. Lines 3-4 are the resulting code of the transformation for attribute `roll` (regular part). Lines 5-10 are the result of the transformation for the random selection part in line 5 of $\Pi_{dice}$. Line 11 is the probabilistic information part, being kept unchanged from the original program (line 6 of $\Pi_{dice}$). Notice that the domain predicates $score/1$ and $dice/1$ were added in some rules (lines 3-10) for variables grounding purpose.

## 3 Examples of Application

In this section we describe several examples that have been tested in P-log(ASP) and P-log(XSB). Several problems are easily solved using probabilistic knowledge, but here we focus on how can inferences be drawn logically.

### 3.1 Cards Problem

Suppose there is a deck of cards divided into spades, hearts, diamonds and clubs. Each suit contains a numbers and pictures. Numbers are from 1 to 10 and pictures are jack, queen and king. What is the probability of having single pair at hand [16]?

```
1. heart   = {h(1..10), h(jack), h(queen), h(king)}.
   spade   = {s(1..10), s(jack), s(queen), s(king)}.
   diamond = {d(1..10), d(jack), d(queen), d(king)}.
   club    = {c(1..10), c(jack), c(queen), c(king)}.
2. cards   = union(heart,spade,diamond,club).
3. number  = {1..5}.
4. draw : number --> cards.
5. random(r(N),draw(N),full):- true.
6. false :- same.
7. same  :- draw(N1,X), draw(N2,X), N1 \= N2.
8. pair(N1,N2):-draw(N1,X1),draw(N2,X2),N1\=N2,sameValue(X1,X2).
9. sameValue(X,Y) :- X =.. [_|V], Y =.. [_|V].
10.singlePair :-
     findall(pair(X,Y), pair(X,Y),[pair(C1,C2),pair(C2,C1)]).
```

**Fig. 1:** Cards Example

The corresponding P-log(XSB) program is shown in Figure 1. Lines 1-4 are declaration of attributes. Line 5 is a random selection rule, saying that the distribution of each attribute $draw_i$ is random. In this example, the same card cannot be drawn twice. This condition is modelled by an ICt in line 6. Lines 8-10 capture the existence of a single pair, namely, predicate $pair(N1, N2)$ in line 8 says that the two different drawings N1-th and N2-th provide a pair, i.e. give the same value (defined by predicate $sameValue/2$ in line 9), and line 10 says that there is only one pair, i.e. there are exactly two different but commutative pairs of drawings that hold (implemented using XSB built-in predicate $findall/3$).

Notice that in this example we use some built-in predicates of XSB that are not supported by ASP such as (=..)/2 and $findall/3$. This feature enables our system to be able to model more complicated problems. To model, for example, the rule for $sameValue/2$ in line 9, in P-log(ASP) we must use a number of rules for grounding that rule. Probability of drawing a single pair is obtained by the query `?- pr(singlePair,Vp)`. The system is supposed to provide the answer [16]: Vp = 0.422569. However, our system only manages to give the answer for the problem with smaller size. Further discussion can be found in subsection 4.2.

### 3.2 Wetgrass Problem

Consider the Wetgrass example [10] in which all nodes are boolean random variables, where the two possible values are denoted by `t`(true) and `f`(false). The corresponding Bayesian Network can be found in Figure 2.

The corresponding P-log(XSB) program is shown in Figure 3. As we see in Figure 2, the event *grass is wet* (W=true) has two possible causes: either the water sprinkler is on (S=true) or it is raining (R=true). The domain declarations are described in lines 1-2. The attributes *sprinkler, rain, cloudy* and *wetgrass* are distributed randomly (line 3). The probabilistic relationships are captured by the Conditional Probability Tables

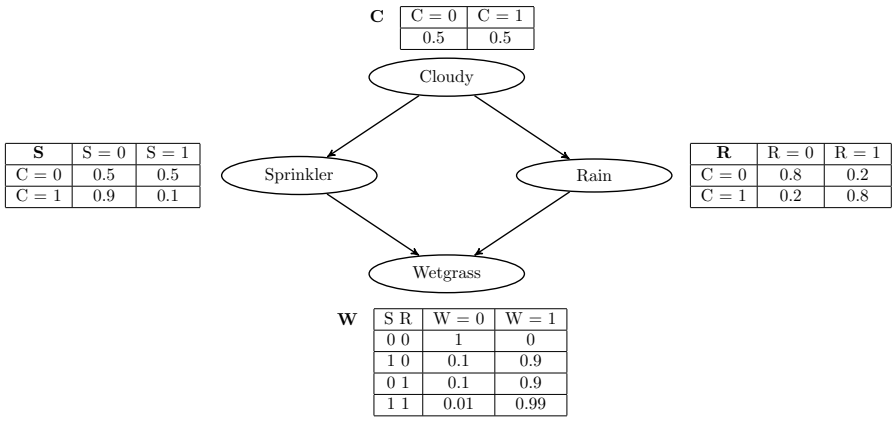


**Fig. 2:** Bayesian Network for Wetgrass Example

(CPTs) in the diagram. The probabilistic information pa-clauses in line 4 represent the conditional probabilities according to the CPTs table.

Suppose we observe that grass is wet. We want to know which one is the more likely cause of wetgrass, raining or sprinkler. The probability of raining being true given wetgrass is true can be easily obtained mathematically $Pr(R = true|W = true) = 0.7079$. Similarly, the probability of sprinkler being on given wetgrass is true: $Pr(S = true|W = true) = 0.4298$. These exact values can be found with the queries `?- pr(rain(t)'|'obs(wetgrass(t)),PR)` and `?- pr(sprinkler(t)'|'obs(wetgrass(t)),PS)` [1], respectively.

**Probabilistic Reasoning with P-log(XSB)** We show how P-log(XSB) can be used as a meta-level language for probabilistic reasoning. The example is extended with the scenario represented in lines 5-9. Imagine that some students have finished their lectures and they are planning to refresh themselves. They have to make decision on one of two choices: going to the beach or the cinema (line 5). If it is not raining, they choose to go to the beach (lines 6–7). Otherwise going to the cinema is the only option (line 8). The probability of raining to be true is higher if it was raining last night (line 9). Based on the observation, this morning they saw the grass was wet. So, what will they choose for their refreshing moment?

This can be done with the query `?-refreshing(X).` The system provides the answer: `X = goingToCinema` since it was raining (line 8), which is based on the fact that the probability of raining given that grass is wet is higher than the one of sprinkler being on under the same condition (line 9).

### 3.3 Random blocks problem [15, 14]

A problem in the random blocks domain consists of a collection of locations, knowledge of which locations left of and below each other, a set of blocks, and knowledge

[1] The probability of A given B is computed using the query `?- pr(A '|' B, V).`

```
1. bool={t,f}.
2. cloudy:bool.  rain:bool.  sprinkler:bool. wetgrass:bool.
3. random(rc, cloudy, full).     random(rr, rain, full).
   random(rs, sprinkler, full).  random(rw, wetgrass, full).
4. pa(rc,cloudy(t),d(1,2)).  pa(rc,cloudy(f),d(1,2)).
   pa(rs,sprinkler(t),d(1,2)) :- cloudy(f).
   pa(rs,sprinkler(t),d(1,10)) :- cloudy(t).
   pa(rr,rain(t),d(2,10)) :- cloudy(f).
   pa(rr,rain(t),d(8,10)) :- cloudy(t).
   pa(rw,wetgrass(t),d(0,1)) :- sprinkler(f),rain(f).
   pa(rw,wetgrass(t),d(9,10)) :- sprinkler(t),rain(f).
   pa(rw,wetgrass(t),d(9,10)) :- sprinkler(f),rain(t).
   pa(rw,wetgrass(t),d(99,100)) :- sprinkler(t),rain(t).
5. refreshing(goingToBeach) :- goingToBeach.
   refreshing(goingToCinema):- goingToCinema.
6. goingToBeach :- sunny.
7. sunny :- not raining.
8. goingToCinema :- raining.
9. raining :- pr(rain(t) '|' obs(wetgrass(t)),PR),
              pr(sprinkler(t)'|'obs(wetgrass(t)),PS), PR > PS.
```

**Fig. 3:** Wetgrass example

of a location for each block. This problem was introduced in [15] to demonstrate the performance of ACE , a system for probabilistic inference based on relational BNs. In [14] it was used as a benchmark to compare latest P-log [12] to ACE (P-log is faster).

We use that problem to compare our system to the latest Plog. The representation of the problem in Plog(XSB) syntax is straightforwardly adapted from the one in P-log(ASP) [14]. Due to lack of space we will not show it here.

## 4 Implementation of the P-log(XSB) system

Our system consists of two main modules: transformation and probabilistic information processing. The first module transforms an original P-log(XSB) program into an appropriate form for further computation by the second module. Both modules were developed on top of XSB Prolog [20], an extensively used and state-of-the-art logic programming inference engine implementation, supporting the Well-Founded Semantics (WFS) for normal logic programs.

The tabling mechanism [19] used by XSB not only provides significant decrease in time complexity of logic program evaluation, but also allows for extending WFS to other non-monotonic semantics. An example of this is the XASP interface (standing for XSB Answer Set Programming) which extends WFS with Smodels to compute stable models [6]. In XASP, only the relevant part to the query of the program is sent to Smodels for evaluation [11]. This allows us to maintain the relevance property for queries over programs, something that ASP does not comply to [7]. ASP obtains all the

complete stable models for the whole program, which might contain redundant information for the evaluation of a particular query. Our approach of using XASP interface sidesteps this issue, sending to Smodels only part of the program that is relevant to the query.

The transformation module transforms the original P-log (XSB) program into a XASP program using five transformation steps described in section 2. This program is then used as the input of the Probabilistic Processing module which will compute all stable models with necessary information for dealing with the query. Only predicates for random attributes and probabilistic information, which have been coded by predicates `pd/2` as the default probability and `pa/3` as the assigned probability are kept in each stable model. In the current version of XASP, those literals are collected by a (posterior) filter after all stable models were generated. This is improved in our system by (ourselves) equipping XASP with a prior filter which enables it to generate stable models with literals by need.

Having obtained stable models with necessary information, the system is ready to answer queries about probabilistic information coded inside the program. Besides queries in form of ASP formulas, our system was extended to be able to answer queries in the form of Prolog predicates which can be defined in a variety of ways. The code for defining the predicate can be included in the original P-log(XSB) program, in a separated XSB prolog program or even asserted into the system. This feature enables us to give very complicated queries which is difficult or even impossible for P-log(ASP) to tackle, e.g. *singlerPair* in the Cards problem. The implementation of this new feature can be done easily with XASP, using the query as a filter for ruling out unsatisfied stable models. In addition, arguing that the system's users usually need to query the program with a number of goals for different kinds of probabilistic information, the system is optimized for answering several queries. In this way the meta-queries illustrated in the previous Wetgrass example can be executed faster, enabling the construction of more sophisticated knowledge bases making use of probabilistic reasoning. Technically, this has been done by memorizing predicates used for processing queries. That means for most of predicates necessary for processing probabilistic queries, we only have to compute them once and reuse the results later without further computation. Moreover, since in the implementation of conditional probability, probabilities of two queries must be computed [1], it benefits even more from the tabling mechanism. The good effects of this choice are clear and discussed below.

### 4.1 Analysis of implementation

Based on the formalisms in section 2 we have successfully implemented P-log in XSB using XASP. In addition, we have compared some computation results of our system to the P-log(ASP) current version [12].

### 4.2 Evaluation

In this section we describe some benchmark problems used to compare the performance of our implementation in XASP with the one of P-log(ASP). We use the same examples described above, just with greater size. For clarity, we denote by $Dice\langle D, X\rangle$ the Dice

example with D dice and each dice having X scores; $Cards\langle D, X\rangle$ – Card example with D drawn cards and X cards on the deck; $Block\langle D, X\rangle$ – Random block example with $D$ blocks and $X$ locations. Since the probability of a formula w.r.t. a program depends on stable models in which it is true and those models can be obtained by combining the ones of its atomic elements, we can use just the set of atomic formulas, for example $roll(d1, i)$ and $roll(d2, i)$ $(i = 1, \ldots, 20)$ in $Dice\langle 2, 20\rangle$, for testing the performance of the systems, reflecting the performance of any formulas.

Table 1 shows the number of stable models, time for the first run, average time from second to tenth runs as well as average time of ten runs of atomic queries (queries for atomic formulas) w.r.t. P-log(XSB) and P-log(ASP) on a computer running Linux Ubuntu 8.10, 1.8 Ghz core 2 dual, and 2 GByte of RAM. The P-log(XSB) system was run on XSB 3.2. Both systems use Smodels 2.33, Lparse-1.1.1. Notice that since we use a set of atomic queries for comparison, only relevant part of the examples which does not contain P-log(ASP) unsupported constructs is necessary for the P-log(ASP) site.

We have run with a number of sets of ten randomly selected atomic queries in both systems and realized that the relative performance of the systems did not depend on the selected sets. This is because of the way the probability of formulas is computed: in any case, the unnormalized probability of every stable model of the program must be computed, and then reused for each stable model in the list of the ones that satisfy the formula. Also note that in our system it is possible to run all ten queries at once, making a conjunctive query with those ten queries in the body, and the performance is identical to the one obtained by running the queries separately. However, this is not the case with P-log(ASP) where the queries have to be run separately. We consider only the total time of computing stable models and deriving the answers. The time of transformation is considerably small on both systems, thus being ignored.

**Table 1:** Benchmark

| | | P-Log(XSB) | | | | P-Log(ASP) | | | |
|---|---|---|---|---|---|---|---|---|---|
| Examples | Number of Stable Models | First Run | Second Run | Average (10 times) | Average (2nd - 10th) | First Run | Second Run | Average (10 times) | Average (2nd - 10th) |
| Dice$\langle 2, 20\rangle$ | 400 | 0.1160 | 0.0120 | 0.0216 | 0.0111 | 0.07 | 0.03 | 0.0456 | 0.0411 |
| Dice$\langle 2, 50\rangle$ | 2,500 | 3.7880 | 0.1000 | 0.4872 | 0.1204 | 0.71 | 0.72 | 0.7189 | 0.72 |
| Dice$\langle 2, 100\rangle$ | 10,000 | 88.2490 | 0.6600 | 9.4181 | 0.6591 | 7.41 | 7.41 | Down in 6th run | |
| Cards$\langle 5, 8\rangle$ | 6,720 | 1.5520 | 0.3360 | 0.3764 | 0.2458 | 0.63 | 0.66 | 0.6367 | 0.6378 |
| Cards$\langle 5, 10\rangle$ | 30,240 | 8.1320 | 1.7000 | 2.1221 | 1.4543 | 3.68 | 3.69 | Down in 7th run | |
| Cards$\langle 5, 11\rangle$ | 55,440 | 15.9440 | 3.2970 | 4.7246 | 3.4780 | 7.64 | 7.52 | Down in 4th run | |
| Cards$\langle 5, 12\rangle$ | 95,040 | 29.3250 | 6.0010 | 8.6161 | 6.3151 | 14.49 | Down | - | - |
| Block$\langle 3, 22\rangle$ | 9,240 | 6.3440 | 0.5000 | 0.9724 | 0.3756 | 1.72 | 1.7 | 1.7033 | 1.7 |
| Block$\langle 3, 30\rangle$ | 24,360 | 24.5290 | 1.6280 | 3.4518 | 1.1099 | 6.39 | 6.37 | Down in 5th run | |
| Block$\langle 3, 35\rangle$ | 39,270 | 48.0030 | 2.9200 | 6.5396 | 1.9326 | 12.37 | 17.88 | Down in 3th run | |
| Block$\langle 3, 40\rangle$ | 59,280 | 89.2610 | 4.9880 | 12.5167 | 3.9896 | 25.92 | Down | - | - |

According to the results, for the first run, our system is about 1.5 to 2 times slower than P-log(ASP), except for $Dice\langle 2, 100\rangle$ where ours is 11 times slower. This is due to the fact that the cardinality constraints of Smodels language have not been available for the current version of XASP (namely, $xnmr\_int$ interface). As discussed in Section 2

we need additional rules for determining upper and lower bounds of the constraints, resulting in that the larger cardinality interval is, the worse the performance of our system is. However, from the second time on, ours is faster, namely, from 3 to 9 times if the first one not being taken into account and from 2 to 5 times otherwise. Therefore, the more probabilistic information we need to extract from the knowledge base, the more useful our system is. Furthermore, P-log(XSB) is more stable (see the cases where P-log(ASP) crashes (denoted by $Down$)).

## 5 Conclusions and Future Work

We have described our approach for re-implementing P-log in XSB using XASP. With XASP, the relevance of the system is maintained, thus Smodels only needs to work with the relevant part and derives only necessary information for further processing. In addition, the tabling mechanism of XSB significantly decreases the computation time of the system by reusing the computations having been done. By comparing the two systems using some benchmarks, we have shown that although our system is slower for the first query, it is much faster for the subsequent queries.

We have extended P-log with new features, first of all, to query the system with more expressive queries not supported in original version. This feature enables users with a very powerful way to gain necessary information from the knowledge base. Furthermore, in many practical problems the domain of one attribute needs to be represented by, e.g. union, intersection, of other domains. Hence, some set operations for domain definition equipped in our system make the users easier in representing those problems.

In our implementation, we have made some important changes in XASP package, namely, $xnmr\_int$ interface, which result in a better performance of the system. We also plan to improve $xnmr\_int$ in order to accept the full lparse syntax, particularly the cardinality constraints. We expect that in this way our system will obtain a comparable performance for the first run.

In general, the approach to probabilistic reasoning by deriving all possible worlds has to deal with a very big list of stable models with a number of predicates. In any cases, we have to compute the unnormalized probability for each stable model of the list. Since the computation can be done in parallel, the performance of the system would very much benefit from multicore CPU computers by using multi-threading, which is very efficient in XSB, from version 3.0 [20]. This approach will be explored in our next version. We also envisage to explore properties of the transformed programs in order to control the exponential blow-up of stable models, w.r.t. some specific cases, e.g. programs resulting from the transformation of poly-tree Bayes Networks.

Last but not least, it is worth mentioning that our system has been successfully integrated in several XSB Prolog systems such as ACORDA [17], Evolution Prospection Agents system [18], for modelling uncertainty in decision making. Those systems employ several kinds of preferences which require probabilistic information. We may prefer one abducible to the other if the first one has greater probability, or one outcome to the other if the probability of the first one to occur is greater than the other one.

## References


1. C. Baral, M. Gelfond, and N. Rushton. *Probabilistic reasoning with answer sets.* In Proceedings of LPNMR7, pages 21−33, 2004.
2. C. Baral, M. Gelfond and N. Rushton. *Probabilistic Reasoning with Answer Sets.* TPLP, Volume 9, Part 1, pages 57-144, January 2009.
3. D. Poole. *Probabilistic horn abduction and bayesian networks.* Artificial Intelligence, 64(1):81-129,1993.
4. D. Poole. *The independent choice logic for modelling multiple agents under uncertainty.* Artificial Intelligence, 94(1-2):5−56, 1997.
5. D. Poole. *Abducing through negation as failure: Stable models within the independent choice logic.* Journal of Logic Programming, 44:535, 2000.
6. I. Niemelã,P. Simons. *Smodels: An implementation of the stable model and well-founded semantics for normal logic programs.* Procs. of LPNMR, Berlin, 1997.
7. J. Dix. *A classification theory of semantics of normal logic programs: i. strong properties, ii. weak properties.* Fundamenta Informaticae, 22(3):227255,257288, 1995.
8. J. Pearl. *Causality.* Cambridge U. P. 2000.
9. J. Vennekens, S. Verbaeten, and M. Bruynooghe. *Logic programs with annotated disjunctions.* In ICLP, pages 431−445, 2004.
10. K. Murphy. *A Brief Introduction to Graphical Models and Bayesian Networks.* http://www.cs.ubc.ca/∼murphyk/Bayes/bayes.html, [Accessed 25 January 2008].
11. L. Castro, T. Swift, and D. S. Warren. *XASP: Answer Set Programming with XSB and Smodels.* http://xsb.sourceforge.net/packages/xasp.pdf
12. Latest P-log implementation (September 2008) downloadable at http://www.cs.ttu.edu/∼wezhu/
13. M. Gelfond, V. Lifschitz. *The stable model semantics for logic programming.* Procs of the 5th Intl. Conf. and Symp. on Logic Programming. 1070–1080, 1988.
14. M. Gelfold, N. Rushton and W. Zhu. *Combining Logical and Probabilistic Reasoning.* AAAI Spring Symposium, 2006.
15. M. Chavira, A. Darwiche, M. Jaeger, *Compiling relational Bayesian networks for exact inference.* Procs. European Workshop on Probabilistic Graphical Models, Netherlands, 2004.
16. M. Shackleford, A.S.A. *Problem 62 Solution - Probabilities in Poker.* http://mathproblems.info/prob62s.htm, [Last accessed 10 June 2008].
17. L. M. Pereira, C. K. Ramli, *Modelling Probabilistic Causation in Decision Making*, Procs. First KES Intl. Symposium on Intelligent Decision Technologies, Springer, Japan, 2009.
18. L. M. Pereira, H. T. Anh. *Evolution Prospection*, Procs. First KES Intl. Symposium on Intelligent Decision Technologies, Springer, Japan, 2009.
19. T. Swift. *Tabling for non-monotonic programming.* Annals of Mathematics and Artificial Intelligence, 25(3–4):201-240, 1999.
20. *The XSB System Version 3.0 Volume 2: Libraries, Interfaces and Packages.* July, 2006.