Probabilistic reasoning with answer sets

Chitta Baral[1], Michael Gelfond[2], and Nelson Rushton[2]

[1] Department of Computer Science and Engineering
Arizona State University
Tempe, Arizona 85287
chitta@asu.edu
[2] Department of Computer Science
Texas Tech University
Lubbock, Texas 79409
mgelfond@cs.ttu.edu, nrushton@coe.ttu.edu ⋆

Abstract. We give a logic programming based account of probability and describe a declarative language P-log capable of reasoning which combines both logical and probabilistic arguments. Several non-trivial examples illustrate the use of P-log for knowledge representation.

1 Introduction

> A man is sitting at a blackjack table, where cards are being dealt from a single deck. What is the probability he is dealt a blackjack (two cards, one of which is an ace, and the other of which is a 10 or a face card)? The standard answer is $4 * 16/C(52, 2)$. Now suppose that on the previous hand, cards removed from the deck were a king, two 3's, an 8 and a 5. This changes the resulting calculation – but only for someone who saw the cards dealt, and takes them into account. Considering more information could change the result even further. In fact, the probability the player receives a blackjack will be either 1 or 0 if we take into account the arrangement of the already-shuffled cards lying in the shoe.

This simple example illustrates an important point: In order to be well posed, questions about probabilities must be asked and answered with respect to a body of knowledge. In this paper we introduce P-log, a language for representing such knowledge. P-log allows the user to represent both logical knowledge and basic probabilistic information about a domain; and its semantics provides a mechanism for systematically deriving conditional and unconditional probabilities from the knowledge represented. P-log uses A-Prolog[3] or its dialects to express logical knowledge. Basic probabilistic information is expressed by probability atoms, say $pr(a|_c\ B) = v$, which is read intuitively as saying *a is caused by factors determined by B with probability v*. As noted in [16],

⋆ We want to thank the reviewers for useful comments. The last two authors were partially supported by NASA under grants NCC9-157, NAG2-1560.

[3] The language of logic programs with classical and default negation and disjunction under the answer set semantics [4].

causal probabilities differ from ordinary conditional probabilities in two respects. First, a causal probability statement implicitly represents a set of conditional independence assumptions: given its cause C, an effect E is probabilistically independent of all factors except the (direct or indirect) effects of E. Second, causal probabilities can be used to determine the effects of actions which interrupt the normal mechanisms of a model, while conditional probabilities cannot do this in general (see Example 4). Both of these differences are captured in the semantics of P-log.

2 The P-log Language

2.1 Syntax of P-log

Let $\mathcal{L}$ be a dialect of A-Prolog (e.g. [13, 14, 3, 2]). A *probabilistic logic program* (P-log program), Π, over $\mathcal{L}$ consists of *sorted signature*, *declarations*, *regular rules* of $\mathcal{L}$, *probabilistic information*, *observations*, and *actions*.

Signature: The sorted signature Σ of Π contains sets O, F, and R of object, function, and relation names respectively. We assume F is the disjoint union of sets F_r and F_a. Members of F_a will be called *attributes*. Terms are formed from O and F_r, and atoms from R and the set of terms. In addition, we allow atoms of the form $a(\bar{t}) = t_0$, where t_0 is a term, $\bar{t}$ a vector of terms, and a is an attribute. Terms and literals are normally denoted by (possibly indexed) letters t and l respectively; letters c's, a's, and r's are used as generic names for sorts, attributes and relations; other lower case letters denote objects; capital letters stand for variables. As usual a rule with variables is viewed as a shorthand for the collection of its ground instances (with variables replaced by properly sorted ground terms).

The **declaration** of a P-log program is a collection of definitions of sorts, and typing information for attributes and relations.

A sort c can be defined by explicitly listing its elements, $c = \{x_1, \ldots, x_n\}$, or by a logic program with a unique answer set A. In the latter case $x \in c$ iff $c(x) \in A$. A statement

$$\text{rel } r_1 : c_{11} \times \ldots \times c_{1n}, \ldots, r_k : c_{k1} \times \ldots \times c_{km} \tag{1}$$

specifies sorts for parameters of relations $r_1, \ldots, r_k$. The domain and range of an attribute a are given by a statement

$$a : c_1 \times \ldots \times c_n \rightarrow c_0 \tag{2}$$

For relations and attributes without parameters we simply write $rel\ r$ and $a : c_0$ respectively.

The following example will be used throughout this section.

Example 1. Consider a domain containing two dice. A P-log program Π_0 modeling the domain will have a signature Σ containing the names of the two dice, d_1 and d_2, an attribute $roll$ mapping each die into its value, an integer from 1 to 6, relations

$owns(D, P)$, $even(D)$, and $even$ where P and D range over the sorts $person$ and $dice$ respectively, and "imported" arithmetic functions $+$ and mod. The corresponding declarations, D_1, will be as follows:

$dice = \{d_1, d_2\}$. $score = \{1, 2, 3, 4, 5, 6\}$. $person = \{mike, john\}$.
$roll : dice \rightarrow score$.
rel $owns : dice \times person,\ even : dice,\ even.$

The **regular part** of a P-log program consists of a collection of rules of $\mathcal{L}$. A rule can contain atoms of the form $a(\bar{t}) = y$ which are viewed as shorthand for an $\mathcal{L}$ atom $a(\bar{t}, y)$). For instance, regular part D_2 of program Π_0 may contain rules of A-Prolog

$even(D) \leftarrow roll(D) = Y, Y\ mod\ 2 = 0.$
$\neg even(D) \leftarrow not\ even(D).$
$even \leftarrow roll(d_1) = Y_1, roll(d_2) = Y_2, (Y_1 + Y_2)\ mod\ 2 = 0.$
$owns(d_1, mike).\ \ owns(d_2, john).$

Probabilistic information consist of statements of the form:

$$random\ \ a(\bar{t})\ :\ B \tag{3}$$

$$pr(a(\bar{t}) = y\ |_c\ B) = v \tag{4}$$

where $v \in [0, 1]$, B is a collections of Σ-literals, and pr is a special symbol not belonging to Σ. By $pr(a(\bar{t}) = y\ |_c\ B)$ we denote the probability of $a(\bar{t}) = y$ *being caused by factors determined by* B. If B in (3) (or (4)) is empty we simply write $random\ \ a(\bar{t})$ (or $pr(a(\bar{t}) = y) = v$). Statement (3) says that, given B, the value of $a(\bar{t})$ is *normally* selected at random; (4) gives a causal probability of a particular selection. For instance, the dice domain may include probabilistic part, D_3:

$random\ \ roll(D).$
$pr(roll(D) = Y\ |_c\ owns(D, john)) = 1/6.$
$pr(roll(D) = 6\ |_c\ owns(D, mike)) = 1/4.$
$pr(roll(D) = Y\ |_c\ Y \neq 6, owns(D, mike)) = 3/20.$

This says that the die owned by John is fair, while the die owned by Mike is biased to roll 6 at a probability of .25. Statements of type (4) will be sometimes referred to as *probabilistic atoms*.

We will have a special agreement for boolean attributes. First, $pr(a(\bar{t}) = true)$ and $pr(a(\bar{t}) = false)$ will be written as $pr(a(\bar{t}))$ and $pr(\neg a(\bar{t}))$. Second, for each probabilistic atom $pr(a(\bar{t})) = v$ from the program we will automatically generate the atom $pr(\neg a(\bar{t})) = 1 - v$. This will allow the user to write fewer probabilistic atoms.

Observations and **actions** are statements of the respective forms

$$obs(l). \qquad do(a(\bar{t}) = y)).$$

Observations are used to record the outcomes of random events, i.e., random attributes and their consequences. The dice domain may, for instance, contain $\{obs(roll(d_1) = 4)\}$ recording the outcome of rolling dice d_1. $do(a(\bar{t}) = y)$ indicates that $a(\bar{t}) = y$ is made true as a result of a deliberate (non-random) action. For instance, $\{do(roll(d_1) = 4)\}$ may indicate that d_1 was simply put on the table in the described position. The meaning of do is briefly discussed in the definition of the semantics and in Examples 3 and 5. For more detailed discussion of the difference between actions and observations in the context of probabilistic reasoning see [16]. The program Π_0 obtained from Π by removing observations and actions will be referred to as the *base* of Π. This program normally contains the reasoner's initial knowledge.

2.2 Semantics of P-log

The semantics of a probabilistic program Π (over dialect $\mathcal{L}$ of A-Prolog) is given by the sets of beliefs of a rational agent associated with Π, together with their probabilities. Sometimes we refer to these sets as possible worlds of Π. The precise definition is based on the mapping of Π into its $\mathcal{L}$ counterpart, Π', describing Π's intuitive meaning.

Construction of Π'

Π' is a program of $\mathcal{L}$ consisting of sort declarations of Π (with $c = \{x_1, \ldots, x_n\}$ interpreted as $c(x_1), \ldots, c(x_n)$), its regular part, actions and observations, and the collection of rules:

1. Random Selection:
   - For each non-boolean attribute a with range $\{y_1, \ldots, y_m\}$:

   $$a(\overline{X}, y_1) \text{ or } \ldots \text{ or } a(\overline{X}, y_m) \leftarrow random(a(\overline{X})) \quad (5)$$

   - For each boolean attribute a:

   $$a(\overline{X}) \text{ or } \neg a(\overline{X}) \leftarrow random(a(\overline{X})) \quad (6)$$

   (Note that in both cases $\overline{X}$ will not be present for attributes of arity 0).
   - For each attribute a:

   $$\neg a(\overline{X}, Y_1) \leftarrow a(\overline{X}, Y_2), Y_1 \neq Y_2, \quad (7)$$

2. Randomness and Deliberate Actions:
   - Random attributes normally have random values: For each declaration (3):

   $$random(a(\overline{t})) \leftarrow B, not\ \neg random(a(\overline{t})) \quad (8)$$

   - Deliberate actions are successful:

   $$a(\overline{X}, Y) \leftarrow do(a(\overline{X}) = Y). \quad (9)$$

   - The value assigned to $a(\overline{X})$ by a deliberate action is not random.

   $$\neg random(a(\overline{X})) \leftarrow do(a(\overline{X}) = Y) \quad (10)$$

   This is a cancellation axiom for (8).

3. Causal Probabilities:
   For each probability atom (4):

$$pr(a(\bar{t}, y), v) \leftarrow B, a(\bar{t}, y), random(a(\bar{t})). \tag{11}$$

   This rule assigns probability v to $a(\bar{t}) = y$ in every possible world in which $a(\bar{t}) = y$ is caused by B.
4. Eliminating Impossible Worlds
   – Accounting for the observations:
     For every attribute a,

$$\leftarrow obs(a(\overline{X}, Y)), not\ a(\overline{X}, Y)). \tag{12}$$

     The rule selects possible worlds which match the observation. Worlds not containing $a(\overline{X}, Y)$ are eliminated.
   – Possible worlds must have non-zero probabilities:

$$\leftarrow pr(a(\overline{X}, Y), 0). \tag{13}$$

     This rule ensures that every possible world of the program is truly possible, i.e., has a non-zero probability.
   – The value of an attribute is either random or is assigned by a deliberate action but not both.

$$\leftarrow obs(a(\overline{X}, Y)), do(a(\overline{X}) = Y). \tag{14}$$

This completes the construction of Π'.

Now we are ready to define the probability measure, μ, induced by the P-log program Π with signature Σ.

Definition 1. (Possible Worlds)
The set, W, of Σ-literals from an answer set of Π' is called a *possible world* (answer set) of Π.

It is not difficult to show that there is one-to-one correspondens, α, between the set Ω of possible worlds of Π and the collection of answer sets of Π'.

Definition 2. (The probability measure)

1. The *unnormalized probability*, $\hat{P}_{\Pi}(W)$, of a possible world W is

$$\hat{P}_{\Pi}(W) = \prod_{pr(l,v) \in \alpha(W)} v$$

2. The *probability measure*, $\mu_{\Pi}(W)$, of a possible world W *induced by* Π is the unnormalized probability of W divided by the sum of the unnormalized probabilities of all possible worlds of Π, i.e.,

$$\mu_{\Pi}(W) = \hat{P}_{\Pi}(W) / \sum_{W_i \in \Omega} \hat{P}_{\Pi}(W_i)$$

Formulas of Σ are constructed from atoms of Σ and the symbol $\top$ using $\wedge$, or , and $\neg$. We follow [18] and expand the probability measure induced by Π to define probability, P_Π of formulas of Σ. First we define the truth ($W \vdash A$) and falsity ($W \dashv A$) of a formula A with respect to a possible world W:

1. $W \vdash \top$.
2. For any Σ-literal l, $W \vdash l$ if $l \in W$; $W \dashv l$ if $\bar{l} \in W$[4]
3. $W \vdash A_1 \wedge A_2$ if $W \vdash A_1$ and $W \vdash A_2$;
   $W \dashv A_1 \wedge A_2$ if $W \dashv A_1$ or $W \dashv A_2$.
4. $W \vdash A_1$ or A_2 if $W \vdash A_1$ or $W \vdash A_2$; $W \dashv A_1$ or A_2 if $W \dashv A_1$ and $W \dashv A_2$.
5. $W \vdash \neg A$ if $W \dashv A$; $W \dashv \neg A$ if $W \vdash A$.

A formula A which is neither true nor false in W is *undefined* in W. Now we are ready for the main definition:

Definition 3. (Probability)
The *probability*, $P_\Pi(A)$, of a formula A, is the sum of the measures of the possible worlds of Π in which A is true, i.e.

$$P_\Pi(A) = \sum_{W \vdash A} \mu(W).$$

At this point it maybe useful to note the number of substantial differences between this definition and a more traditional definition of (subjective) probability from [18]. We share the view from [18] that

> ... probability is an agent's measure of belief in some propositions based on the agent's knowledge.

Moreover, both approaches use the language of possible worlds and probability measures. But the possible worlds of the latter are two-valued interpretations describing possible states of the world; the agent's knowledge is ultimately given by a probabilistic measure which can be directly defined or extracted from conditional probabilities and independence assumptions. The former defines possible worlds as beliefs of the rational agent whose background knowledge is represented by program Π; the corresponding measure is induced by causal probabilistic information from the program.

Explicit representation of agent's beliefs allow us to give a natural formalization of the classical notion of conditional probability $P_{Pi}(A|C)$, where A is a formula of Σ and C consists of σ-literals and actions.

Definition 4. (Conditional Probability)
The *conditional probability*, $P_\Pi(A|C)$ is the probability $P_R(A)$ where $R = \Pi \cup C^*$ and $C^* = \{obs(l) : l \in C\} \cup \{do(l) : do(l) \in C\}$.

(The choice of this definition as opposed to a more traditional definition of conditional probability will be discussed in the following sections.)

[4] $\bar{l} = \neg p(\bar{t})$ if $l = p(\bar{t})$; $\bar{l} = p(\bar{t})$ if $l = \neg p(\bar{t})$.

Definition 5. A probabilistic program Π is said to be *consistent* if

1. Π' is consistent (i.e., has a consistent answer set).
2. Let Π_0 be the base of Π. Then, for any probability atom $pr(l|_c B) = y$ from Π_0, the conditional probability $P_{\Pi_0}(l|B) = y$ whenever the latter is defined.
3. Whenever $pr(l|B_1) = y_1$ and $pr(l|B_2) = y_2$ belong to Π, no possible world of Π satisfies B_1 and B_2.

The first requirement ensures the consistency of the program rules. The second guarantees that P_Π satisfies the probabilistic statements from Π. The third requirement enforces the independence assumptions embodied in causal probabilities: given its cause B, an effect l has a fixed probability, independent of all other factors (except for the effects of l).

The following proposition says that P_Π satisfies axioms of probability.

Proposition 1. For consistent P-log program Π:

1. For every formula A, $0 \leq P_\Pi(A) \leq 1$,
2. $P_\Pi(\top) = 1$, and
3. $P_\Pi(A \text{ or } B) = P_\Pi(A) + P_\Pi(B)$, for any mutually exclusive formulas A and B.

(Note that, since A or $\neg A$ may be undefined in a possible world W, $P_\Pi(A \text{ or } \neg A)$ is not necessarily equal to 1. This explains our use of $\top$ in the clause (2) above.)

To illustrate these definitions let us further elaborate the "dice" example.

Example 2. Let T_0 consist of first three sections D_1, D_2, D_3, of the "dice" program from Example 1. Then T_0' consists of rules of D_2 and the rules:

$dice(d1).\ dice(d2).\ person(mike).\ person(john).$
$score(1).\ score(2).\ score(3).\ score(4).\ score(5).\ score(6).$
$roll(D,1)\ or\ roll(D,2)\ or\ \ldots\ or\ roll(D,5)\ or\ roll(D,6) \leftarrow random(roll(D)).$
$\neg roll(D,Y_2) \leftarrow roll(D,Y_1), Y_1 \neq Y_2.$
$random(roll(D)) \leftarrow not\ \neg random(roll(D)).$
$\neg random(roll(D)) \leftarrow do(roll(D) = Y).$
$pr(roll(D,Y), 1/6) \leftarrow owns(D, john), roll(D,Y), random(roll(D)).$
$pr(roll(D,6), 1/4) \leftarrow owns(D, mike), roll(D,6), random(roll(D)).$
$pr(roll(D,Y), 3/20) \leftarrow Y \neq 6, owns(D, mike), roll(D,Y), random(roll(D)).$
$\leftarrow obs(a(X,Y)), not\ a(X,Y).$
$a(X,Y) \leftarrow do(a(X,Y).$
$\leftarrow pr(a(X,Y), 0).$

It is easy to check that T_0' has 36 answer sets containing different pairs of atoms $roll(d_1, i_1)$ and $roll(d_2, i_2)$. Each answer set of T_0' containing $roll(d_1, 6)$ will contain a probability atom $pr(roll(d_1, 6), 1/4)$, as well as a probability atom $pr(roll(d_2, i), 1/6)$ for some i, and hence have the probability $1/24$. Any other answer set has probability $1/40$. It is easy to check that the program is consistent.

Now let $T_1 = T_0 \cup \{obs(roll(d_1, 4))\}$. By definition,
$P_{T_0}(even|roll(d_1, 4)) = P_{T_1}(even) = 1/2$. The same result can be obtained by using classical definition of conditional probability,

$$P(A|B) = P(A \wedge B)/P(B) \tag{15}$$

The following proposition shows that this is not a coincidence.

Proposition 2. let Π be a consistent P-log program over the A-Prolog. Then for every formula A and every collection, B, of Σ-literals such that $P_\Pi(B) \neq 0$, P_Π satisfies condition (15) above.

Note that the asymmetry between A and B is not essential. It is caused by the syntactic restriction of A-Prolog which prohibits arbitrary formulas in the heads of rules. The restriction can be lifted if instead of A-Prolog we consider its dialect from [8] which can be done with very minor changes in the proof.

Example 3. Consider a program, P_0

$random\ a : boolean.$
$pr(a) = 1.$

Recall that P_0 will be (automatically) expanded to include a new probability atom, $pr(\neg a) = 0$. It is easy to see that P_0' has one answer set, which contains a (the possible answer set containing $\neg a$ is eliminated by constraint (13)). Obviously, $P_{P_0}(a) = 1$ and hence the program is consistent. Now we compare P_0 with the following program P_1:

$random\ a : boolean.$
$a.$

The programs have the same possible worlds and the same probability measures. However, they express different information. To see that, consider programs P_2 and P_3 obtained by adding the statement $do(\neg a)$ to P_0 and P_1 respectively. P_2 remains consistent — it has one possible world $\{\neg a\}$ — while P_3 becomes inconsistent (see rule (9)). The statement $pr(a) = 1$ is defeasible while the statement a is not. This does not mean however that the former can be simply replaced by the corresponding default. To see that consider Π_4

$random\ a : boolean.$
$a \leftarrow not\ \neg a.$

Π_4 has two possible worlds, $\{a\}$ and $\{\neg a\}$ (note the interplay between the default and rule 6 of Π_4'). In other words "randomness" undermines the default. Finally consider programs P_5:

$random\ a : boolean.$
$a.$
$pr(a) = 1/2.$

and P_6:

$random\ a : \{0, 1, 2\}.$
$pr(a = 0) = pr(a = 1) = pr(a = 2) = 1/2.$

Both programs are inconsistent. P_5 has one possible world $W = \{a\}$, but $\hat{P}_{P_5}(W) = 1/2$, and hence $P_{P_5}(a) = 1$ instead of $1/2$.

P_6 has three possible worlds, $\{a(0), \neg a(1), \neg a(2)\}$, $\{\neg a(0), a(1), \neg a(2)\}$, and $\{\neg a(0), \neg a(1), a(2)\}$ each with unnormalized probability $1/2$. Hence $P_{P_6}(a(0)) = 1/3$ instead of $1/2$. (Let $V(B, \bar{t})$ be a multiset of v such that $pr(a(\bar{t} = y) = v \in \Pi$ for some $y \in range(a)$. Then it can be shown that if Π is consistent then for every B and $\bar{t}$ the sum of the values in $V(B, \bar{t})$ is 1).

3 Representing knowledge in P-log

Now we give several examples of non-trivial probabilistic knowledge representation and reasoning performed in P-log.

Example 4. (Monty Hall Problem)
We start with solving the Monty Hall Problem, which gets its name from the TV game show hosted by Monty Hall (we follow the description from http://www.io.com/∼kmellis/monty.html): A player is given the opportunity to select one of three closed doors, behind one of which there is a prize. The other two rooms are empty. Once the player has made a selection, Monty is obligated to open one of the remaining closed doors, revealing that it does not contain the prize. He then asks the player if he would like to switch his selection to the other unopened door, or stay with his original choice. Here is the problem: Does it matter if he switches?

The answer is YES. In fact switching doubles the player's chance to win. This problem is quite interesting, because the answer is felt by most people — including mathematicians — to be counter-intuitive. Most people almost immediately come up with a (wrong) negative answer and not easily persuaded that they made a mistake. We believe that part of the reason for the difficulty is some disconnect between modeling probabilistic and non-probabilistic knowledge about the problem. In P-log this disconnect disappears which leads to a natural correct solution. In other words, the standard probability formalisms lack the ability to formally represent certain non-probabilistic knowledge that is needed in solving this problem. In the absence of this knowledge, wrong conclusions are made. We will show that the use of P-log avoids this, as P-log allows us to specify this knowledge explicitly.

The domain contains the set of three doors and three 0-arity attributes, $selected$, $open$ and $prize$. This will be represented by the following P-log declarations:

1. $doors = \{1, 2, 3\}$.
2. $open, selected, prize : doors$.

The regular rule section states that Monty can only open a door to a room which is not selected and which does not contain the prize.

3. $\neg can\_open(D) \leftarrow\ selected = D$.
4. $\neg can\_open(D) \leftarrow\ prize = D$.
5. $can\_open(D) \leftarrow\ not\ \neg can\_open(D)$.
6. $\leftarrow\ open = D, \neg can\_open(D)$.

This knowledge (which can be extracted from the specification of the problem) is often not explicitly represented in probabilistic formalisms leading to reasoners (who usually do not realize this) to insist that their wrong answer is actually correct.

The probabilistic information about the three attributes of doors can be now expressed as follows:

7. $random\ prize, selected, open.$
8. $pr(prize = D) = 1/3.$
9. $pr(selected = D) = 1/3.$
10. $pr(open = D \mid_c can\_open(D), can\_open(D_1), D \neq D_1) = 1/2.$

The last rule is where most reasoners make a mistake. They assume that the probability that Monty opens one of the remaining doors is 1/2. That is not the case. Monty knows which door has a prize. If the prize is behind one of the unopened doors, he is not going to open that one. In that case the probability of opening the door which has the prize is 0 and the probability for the other one is 1. On the other hand if both unselected doors do not have the prize, then and only then can Monty open either of the door with probability 1/2. The above information is elegantly expressible in P-log and most standard probabilistic reasoning language can not express it, without falling back on a natural language such as English.

To eliminate an orthogonal problem of modeling time we assume that the player has already selected door 1, and Monty opened door 2.

$obs(selected = 1).\ \ obs(open = 2).\ \ obs(\neg prize = 2).$

Let us refer to the above P-log program as M. Because of the observations M has two possible worlds W_1, and W_2: one containing $prize = 1$ and another containing $prize = 3$.

Both W_1 and W_2 contain the probabilistic atom $pr(selected = 1) = 1/3)$. In addition, W_1 contains $pr(prize = 3) = 1/3$ and $pr(open = 2) = 1$ — with the prize being behind door 3 Monty is forced to open door 2. $\hat{P}(W_1) = 1/9$. Similarly, W_2 contains $pr(prize = 1) = 1/3$ and $pr(open = 2) = 1/2$. $\hat{P}(W_2) = 1/18$. This time the prize is behind door 1, i.e. Monty had a choice. Thus $P_M(prize = 3) = 2/3$, and $P_M(prize = 1) = 1/3$. Changing the door doubles the player's chance to win.

Now if the player assumes (either consciously or without consciously realizing it) that Monty could have opened any one of the unopened doors (including one which contains the prize) then his regular rule section will have a different constraint, i.e. (6) will be replaced by:

$\leftarrow\ open = D, selected = D.$

and the rule (10) in his probabilistic part will instead be:

$pr(open = D \mid_c \neg selected = D) = 1/2.$

The resulting program N will also have two possible worlds containing $prize = 1$ and $prize = 3$ respectively, each with unnormalized probability of 1/18, and therefore

$P_N(prize = 1) = 1/2$ and $P_N(prize = 3) = 1/2$. Changing the door will not increase the probability of getting the prize.

It may also be worth mentioning that we do not need to assume that Monty is equally likely to select one of the two available doors. A simple computation shows that the change increases the players chances to win as long as each of the doors can be open with some positive probability. The problem can of course be generalized to an arbitrary number of doors. This can be done for instance by replacing rule (10) by

$pr(open = D \mid_c can\_open(D)) = 1/|\{X : can\_open(X)\}|$

An expression $|\{X : can\_open(X)\}|$ stands for the cardinality of the set of doors Monty can open. (This can be directly encoded in A-Prolog with aggregates.) □

The next example illustrates the ability of P-log to represent and reason with Bayesian networks and to properly distinguish between observations and actions.

Example 5. (Simpson's Paradox)
Let us consider the following story from [16]: A patient is thinking about trying an experimental drug and decides to consult a doctor. The doctor has tables of the recovery rates that have been observed among males and females, taking and not taking the drug.

```
Males:     recover   -recover  num_of_people recovery_rate
 drug        18         12          30            60%
-drug         7          3          10            70%

Females:   recover   -recover  num_of_people recovery_rate
 drug         2          8          10            20%
-drug         9         21          30            30%
```

What should the doctor's advice be? Assuming that the patient is a male, the doctor may attempt to reduce the problem to checking the following inequality

$$P(recover|male, drug) \ > \ P(recover|male, \neg drug) \qquad (16)$$

The corresponding probabilities, given by the tables, are 0.6 and 0.7. The inequality fails, and hence the advice is not to take the drug. This, indeed, is the correct advice. A similar argument shows that a female patient should not take the drug.

But what should the doctor do if he has forgotten to ask the patient's sex? Following the same reasoning, the doctor might check whether

$$P(recover|drug) \ > \ P(recover|\neg drug) \qquad (17)$$

This will lead to an unexpected result. $P(recovery|drug) = 0.5$ while $P(recovery|\neg drug) = 0.4$. The drug seems to be beneficial to patients of unknown sex — though similar reasoning has shown that the drug is harmful to the patients of known sex, whether they are male or female!

This phenomenon is known as Simpson's Paradox: conditioning on A may increase the probability of B among the general population, while decreasing the probability of B in every subpopulation (or vice-versa). In the current context, the important and perhaps surprising lesson is that conditional probabilities do not faithfully formalize what we really want to know: *what will happen if we do X?* In [16] Pearl suggests a solution to this problem in which the effect of deliberate action A on condition C is represented by $P(C|do(A))$ — a quantity defined in terms of graphs describing causal relations between variables. Correct reasoning therefore should be based on evaluating the inequality

$$P(recover|do(drug)) > P(recover|do(\neg drug)) \tag{18}$$

instead of (17) (similarly for (16)). In Pearl's calculus the first value equals $.4$, the second, $.5$. The drug is harmful for the general population as well.

Note that in our formalism $P_\Pi(C|do(A))$ is defined simply as $P_R(C)$ where $R = \Pi \cup \{do(A)\}$ and hence P-log allows us to directly represent this type of reasoning. We follow [16] and assume that the tables, together with our intuition about the direction of causality between the variables, provide us with the values of the following causal probabilities:

$pr(male) = 0.5$, $pr(recover|_c male, drug) = 0.6$,
$pr(recover|_c male, \neg drug) = 0.7$, $pr(recover|_c \neg male, drug) = 0.2$,
$pr(recover|_c \neg male, \neg drug) = 0.3$, $pr(drug|_c male) = 0.75$,
$pr(drug|_c \neg male) = .25$.

These statements, together with declarations:

$random\ \ male, recover, drug\ :\ boolean$

constitute a probabilistic logic program, Π, formalizing the story. The program describes eight possible worlds containing various values of the attributes. Each world is assigned a proper probability value, e.g. $P_\Pi(\{male, recover, drug\}) = .5 * .6 * .75 = 0.225$. It is not difficult to check that the program is consistent. The values of $P_\Pi(recover|_c do(drug)) = .4$ and $P_\Pi(recover|_c do(\neg drug)) = .5$ can be computed by finding $P_{\Pi_1}(recover)$ and $P_{\Pi_2}(recover)$, where $\Pi_1 = \Pi \cup \{do(drug)\}$ and $\Pi_2 = \Pi \cup \{do(\neg drug)\}$. $\Box$

Now we consider several reasoning problems associated with the behavior of a malfunctioning robot. The original version, not containing probabilistic reasoning, first appeared in [6] where the authors discuss the difficulties of solving the problem in Situation Calculus.

Example 6. (A malfunctioning robot)

There are rooms, r_0, r_1, and r_2, reachable from the current position of a robot. The robot navigation is usually successful. However, a malfunction can cause the robot to go off course and enter any one of the rooms. The doors to the rooms can be open or closed. The robot cannot open the doors.

The authors of [6] wanted to be able to use the corresponding formalization for correctly answering simple questions about the robot's behavior including the following "typical" scenario: The robot moved toward open room r_1 but found itself in some other room. What room can this be?

The initial story contains no probabilistic information so we start with formalizing this knowledge in A-Prolog. First we need sorts for time-steps and rooms. (Initial and final moments of time suffice for our purpose).

$time = \{0, 1\}$. $rooms = \{r_0, r_1, r_2\}$.

In what follows we use variable T for time and R for rooms. There will be two actions:

$enter(T, R)$ - the robot *attempts* to enter the room R at time step T.

$break(T)$ - an exogenous breaking action which may alter the outcome of this attempt.

A state of the domain is modeled by two time-dependent relations $open(R, T)$ (room R is opened at moment T), $broken(T)$ (robot is malfunctioning at T), and the attribute, $in(T) : time \rightarrow rooms$, which gives the location of the robot at T.

The description of dynamic behavior of the system is given by A-Prolog rules:

Dynamic causal laws describe direct effects of the actions (note that the last law is non-deterministic):

$broken(T + 1) \leftarrow break(T)$.

$in(T + 1, R) \leftarrow enter(T, R), \neg broken(T + 1)$.

$in(T + 1, r_0)$ or $in(T + 1, r_1)$ or $in(T + 1, r_2) \leftarrow broken(T), enter(T, R)$.

To specify that the robot cannot go through the closed doors we use a constraint:

$\leftarrow \neg in(T, R), in(T + 1, R), \neg open(R, T)$.

Moreover, the robot will not even attempt to enter the room if its door is closed.

$\leftarrow enter(T, R), \neg open(R, T)$.

To indicate that in is a function we use static causal law:

$\neg in(T, R_2) \leftarrow in(T, R_1), R_1 \neq R_2$.

We also need the inertia axioms:

$in(T + 1, R) \leftarrow in(T, R), not\ \neg in(T + 1, R)$.
$broken(T + 1) \leftarrow broken(T), not\ \neg broken(T + 1)$.
$\neg broken(T + 1) \leftarrow \neg broken(T), not\ broken(T + 1)$.
(Similarly for $open$).

Finally, we describe the initial situation:

$open(R, 0) \leftarrow not\ \neg open(R, 0)$.
$in(0, r_1)$.
$\neg in(0, R) \leftarrow not\ in(0, R)$.
$\neg broken(T) \leftarrow not\ broken(T)$.

The resulting program, Π_0, completes the first stage of our formalization.

It is easy to check that $\Pi_0 \cup \{enter(0, r_0)\}$ has one answer set, A, and that $in(1, r_0) \in A$. Program $\Pi_0 \cup \{enter(0, r_0), break(0)\}$ has three answer sets containing $in(1, r_0)$, $in(1, r_1)$, and $in(1, r_2)$ respectively. If, in addition, we are given $\neg open(r_2, 0)$ the third possibility will disappear.

Now we show how this program can be extended by probabilistic information and how this information can be used together with regular A-Prolog reasoning.

Consider Π_1 obtained from Π_0 by adding

random $in(T + 1) : enter(T, R), broken(T + 1)$.
$pr(in(T + 1) = R|_c\ enter(T, R), broken(T + 1)) = 1/2$.
$pr(in(T + 1) = R_1|_c\ R_1 \neq R_2, enter(T, R_2), broken(T + 1)) = 1/4$.

together with the corresponding declarations, e.g.

$in : time \rightarrow rooms$.

It is not difficult to check that probabilistic program $T_1 = \Pi_1 \cup \{enter(0, r_0)\}$ has the unique possible world which contains $in(1) = r_0$. Hence, $P_{T_1}(in(1) = r_0) = 1$. It is easy to show that Π_1 is consistent. (Note that the conditional probabilities corresponding to the probability atoms of Π_1, e.g., $P_{T_1}(in(1) = r_0)|broken(1))$, are undefined and hence (2) of the definition of consistency is satisfied.)

The program $T_2 = T_1 \cup \{break(0)\}$ has three possible worlds — W_0 containing $in(1) = r_0$, and W_1, W_2 containing $in(1) = r_1$ and $in(1) = r_2$ respectively; $P_{T_2}(W_0) = 1/2$ while $P_{T_2}(W_1) = P_{T_2}(W_2) = 1/4$. It is easy to see that T_2 is consistent. Note that $P_{T_1}(in(1) = r_0) = 1$ while $P_{T_2}(in(1) = r_0) = 1/2$ and hence the *additional information changed the degree of reasoner's belief*. □

So far our probabilistic programs were based on A-Prolog. The next example shows the use of P-log programs over CR-Prolog [2] — an extension of A-Prolog which combines regular answer set reasoning with abduction. In addition to regular rules of A-Prolog the new language allows so called *consistency-restoring* rules, i.e., rules of the form

$$l \stackrel{+}{\leftarrow} B \tag{19}$$

The rule says that, given B, the reasoner may believe l but only if l is needed to restore consistency of the program. A rule

$$l \leftarrow B \tag{20}$$

will be called the *strict counterpart* of (19). Let $\Pi = \Pi_0 \cup \Pi_1$ where Π_0 and Π_1 are sets of regular and consistency restoring rules respectively be a program of CR-Prolog. By $sc(\Pi_1)$ we denote the set of strict counterparts of rules from Π_1. An answer set of Π is an answer set of $\Pi_0 \cup R$ where R is minimal (with respect to set inclusion) subset of $sc(\Pi_1)$ such that $\Pi_0 \cup R$ is consistent (i.e. has an answer set). Consider for instance a program, E_1,

1. $p(a) \leftarrow not\ q(a)$.
2. $q(a) \stackrel{+}{\leftarrow}$. 3. $r(a) \stackrel{+}{\leftarrow}$.

It is easy to see that it has one answer set - $\{p(a\}$. in this case the last two rules are not applicable. Suppose now E_1 is expended by a new information, $\neg p(a)$. The new program, $E_2 = E_1 \cup \{\neg p(a)\}$, also has one answer set, $\{\neg p(a), q(a)\}$. This time cr-rule (2) was necessary to restore consistency of the agent's beliefs.

The next example elaborates the initial formalization of the robot story in CR-Prolog.

Example 7. (Probabilistic programs over CR-Prolog)
Let us expand the program T_1 from Example 6 by a CR-rule

$$break(T) \stackrel{+}{\leftarrow} \tag{21}$$

Denote the new program by T_3. The rule (21) says that even though the malfunctioning is rare it may happen. According to the semantics of CR-Prolog this rare event can be used by the agent to ensure consistency of its beliefs.

The semantics of CR-Prolog guarantees that for any collection I of atoms such that $T_1 \cup I$ is consistent, programs $T_1 \cup I$ and $T_3 \cup I$ have the same answer sets; i.e., the conclusions we made so far about the domain will not change if we use T_3 instead of T_1. The added power of T_3 will be seen when the use of T_1 leads to inconsistency. Consider for instance the scenario $I_0 = \{obs(\neg in(1) = r_0)\}$. The first formalization could not deal with this situation — the corresponding program would be inconsistent.

The program $T_4 = T_3 \cup I_0$ will use the CR-rule (21) to conclude $break(0)$, which can be viewed as a diagnosis for an unexpected observation. T_4 has two answer sets containing $in(1) = r_1$ and $in(1) = r_2$ respectively. It is not difficult to check that $P_{T_3}(in(1) = r_0) = 1$ while $P_{T_3}(in(1) = r_0)|I_0) = P_{T_4}(in(1) = r_0) = 0$. Interestingly, *this phenomenon cannot be modeled using classical conditional probabilities*, since classically whenever $P(A) = 1$, the value of $P(A|B)$ is either 1 or undefined.

Our last example will show how Π_1 can be modified to introduce some additional probabilistic information and used to obtain most likely diagnoses.

Example 8. (Doing the diagnostics)
Suppose we are given a list of mutually exclusive faults which could be caused by the breaking action, together with the probabilities of these faults. This information can be incorporated in our program, Π_1, by adding

$faults = \{f_0, f_1\}. \quad fault : time \rightarrow faults.$
$random\ fault(T + 1) : break(T).$
$pr(fault(T) = f_0|_c broken(T)) = .4 \quad pr(fault(T) = f_1|_c broken(T)) = .6$

Let us also assume that chances of the malfunctioning robot to get to room R are determined by the type of the faults, e.g.

$pr(in(1) = r_0|_c fault(1) = f_0) = .2 \quad pr(in(1) = r_0|_c fault(1) = f_1) = .1$
$pr(in(1) = r_1|_c fault(1) = f_0) = .6 \quad pr(in(1) = r_1|_c fault(1) = f_1) = .5,$
$pr(in(1) = r_2|_c fault(1) = f_0) = .2 \quad pr(in(1) = r_2|_c fault(1) = f_1) = .4$

Note that this information supersedes our previous knowledge about the probabilities of in and hence should replace the probabilistic atoms of Π_1. The resulting program, Π_2, used together with $\{enter(0, r_0), obs(\neg in(0) = r_0)\}$ has four answer sets weighted by probabilities. Simple computation shows that at moment 1 the robot is most likely to be in room r_1.

4 Relationship to Existing Work

Our work was greatly influenced by J. Pearl's view on causality and probability. It can be shown that the Bayesian Networks and Probabilistic Causal Models of Pearl can be mapped into P-log programs of similar size. (Proofs of the corresponding theorems will be given in the full version of this paper.) The examples discussed above show that, in addition, P-log allows natural combination of logical and probabilistic information. We were influenced to a lesser degree, by various work incorporating probability in logic programming [12, 10, 7, 11, 9]. In part this is due to our use of answer set semantics, which introduces unique challenges (as well as benefits, in our opinion) for the integration of probabilities.

The closest to our approach is that of Poole [20, 19]. We note three major differences between our work and the work of Poole [20, 19]. First, A-Prolog provides a richer logical framework than does choice logic, including default and classical negation and disjunction. Moreover, our approach works, without modification, with various extensions of A-Prolog including the use of CR-rules. Second, in contrast to Poole's system, the logical aspects of P-log do not "ride on top" of the mechanism for generating probabilities: we bring to bear the power of answer set programming, not only in describing the consequences of random events, but also in the description of the underlying probabilistic mechanisms. Third, our formalization allows the distinction between observations and actions (i.e., doing) to be expressed in a natural way, which is not addressed in choice logic.

There are three elements which, to our knowledge, are new to this work. First, rather than using classical probability spaces in the semantics of P-log, we define probabilities of formulas directly in terms of the answer set semantics. In this way, A P-log program *induces* a classical probability measure on possible worlds by its construction, rather than relying on the existence of a classical measure compatible with it. We see several advantages to our re-definition of probabilities. Most notably, the definition of conditional probability becomes more natural, as well as more general (see Example 7). Also, possible worlds and events correspond more intuitively to answer sets and formulas than to the sample points and random events (i.e., sets of sample points) of the classical theory.

Second, P-log allows us to *elaborate on defaults by adding probabilities* as in Examples 7-8. Preferences among explanations, in the form of defaults, are often more easily available from domain experts than are numerical probabilities. In some cases, we may want to move from the former to the latter as we acquire more information. P-log allows us to represent defaults, and later integrate numerical probabilities by adding to our existing program rather than modifying it. Finally, the semantics of P-log over CR-Prolog gives rise to a unique phenomenon: we can move from one classical probability measure to another merely by adding observations to our knowledge base, as in Example 7. This implies that P-log probability measures are more general than classical ones, since the measure associated with a single P-log program can, through conditioning, address situations that would require multiple distinct probability spaces in the classical setup.

References

1. F. Bacchus. Representing and reasoning with uncertain knowledge. MIT Press, 1990
2. M. Balduccini and M. Gelfond. Logic Programs with Consistency-Restoring Rules. In AAAI Spring 2003 Symposium, 2003.
3. T. Eiter, W. Faber, N. Leone, and G. Pfeifer. Declarative problem solving in dlv. In J. Minker, editor, *Logic Based Artificial Intelligence*, pages 79–103. Kluwer Academic publisher, 2000.
4. M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. In *New Generation Computing*, Vol. 9, NOS. 3,4 365–387, 1991.
5. L. Getoor, N. Friedman, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Relational data mining*, pages 307–335. Springer, 2001.
6. G. Iwan and G. Lakemeyer. What observations really tell us. In *CogRob'02*, 2002.
7. Kristian Kersting and Luc De Raedt. Bayesian logic programs. In J. Cussens and A. Frisch, editors, *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, pages 138–155, 2000.
8. V. Lifschitz, L. R. Tang, and H. Turner. Nested expressions in logic programs. in Annals of Mathematics and Artificial Intelligence, Vol. 25, pp. 369-389, 1999.
9. T. Lukasiewicz. Probabilistic logic programming. In *ECAI*, pages 388–392, 1998.
10. S. Muggleton. Stochastic logic programs. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, page 29. Department of Computer Science, Katholieke Universiteit Leuven, 1995.
11. Raymond T. Ng and V. S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, 1992.
12. Liem Ngo and Peter Haddawy. Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 171(1–2):147–177, 1997.
13. I. Niemelä and P. Simons. Smodels – an implementation of the stable model and well-founded semantics for normal logic programs. In J. Dix, U. Furbach, and A. Nerode, editors, *Proc. 4th international conference on Logic programming and non-monotonic reasoning*, pages 420–429. Springer, 1997.
14. P. Simons,I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence Journal*,138:181–234, 2002/
15. J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, 1988.
16. J. Pearl. *Causality*. Cambridge University Press, 2000.
17. D. Poole. The independent choice logic for modeling multiple agents under uncertainty. *Artificial Intelligence*, 94(1-2 (special issue on economic principles of multi-agent systems)):7–56, 1997.
18. D. Poole,A. Mackworth, and R. Goebel *Computational Intelligence* Oxford University Press, 1998.
19. D. Poole. Abducing through negation as failure: Stable models within the independent choice logic. *Journal of Logic Programming*, 44:5–35, 2000.
20. David Poole. Probabilistic horn abduction and Bayesian networks. *Artificial Intelligence*, 64(1):81–129, 1993.