Refining and Generalizing P-log – Preliminary Report

Evgenii Balai and Michael Gelfond

Texas Tech University, Lubbock, TX, USA
{evgenii.balai,michael.gelfond}@ttu.edu

Abstract. This paper is a preliminary report on the development of a new, improved version of the knowledge representation language P-log. This version clarifies syntax and semantics of the original language and brings informal reading of its main concepts closer to their formal semantics. It also expands P-log with a sophisticated type system and removes some unnecessary restrictions on the occurrences of atoms expressing observations, interventions, and randomness in P-log rules. The generalization simplifies the syntax and allows to formalize reasoning the authors were not able to formalize in the original P-log.

1 Introduction

The paper is a preliminary report on the development of a new, improved version of the knowledge representation language P-log. The language, introduced in [7, 6], is capable of combining non-monotonic logical reasoning about agent's beliefs in the style of Answer Set Prolog (ASP) [9] and probabilistic reasoning with Causal Bayesian Networks [15]. In addition to being logically non-monotonic, P-log is also "probabilistically non-monotonic" - addition of new information can add new possible worlds and substantially change the original probabilistic model. Another distinctive feature of P-log is its ability to reason with a broad range of updates. In addition to standard conditioning on observations, P-log allows conditioning on rules, including defaults and rules introducing new terms, deliberate actions in the sense of Pearl, etc. These and several other features make P-log representations of non-trivial probabilistic scenarios (including such classical "puzzles" as Simpson Paradox, Monty Hall Problem, etc.) very close to their English descriptions which greatly facilitate a difficult task of producing their probabilistic models and sheds some light on subtle issues involved in their solutions. The ability of P-log to represent causal and counterfactual reasoning is discussed in [8]. In [10] P-log was expanded to allow abduction in the style of CR-Prolog [5] and infinite possible worlds. A P-log prototype query answering system (based on ASP solvers) is described in the dissertation [20]. The system allowed the use of P-log in a number of applications such as a system for finding best probabilistic diagnosis for certain types of failure in the Space Shuttle [20], development and automation of mathematical models of machine ethics [16], methods for combining probabilistic and logical reasoning in robotics [19], etc.

Syntactically, P-log is a sorted language whose programs contain definitions of sorts (e.g. $tests = \{t_1, t_2\}$ and $grades = \{a, b, c, d, f\}$) and declarations of functions (e.g. $grade : tests \rightarrow grades$). The atoms of P-log are properly typed expressions of the form $f(\bar{t}) = y$ (or regular arithmetic atoms); literals are atoms or their negations (written as $f(\bar{t}) \neq y$); regular rules are of the form $head \leftarrow body$ where $head$ is a literal and $body$ is a collection of literals possibly preceded by a default negation **not**. In addition P-log allows *random selection rules*

$$random(f(\bar{x}) : \{X : p(X)\}) \leftarrow body \tag{1}$$

describing possible outcomes of random experiments. The rule says that normally the values of $f(\bar{x})$ are selected at random from the set $\{X : p(X)\} \cap \{y_1, \ldots, y_k\}$ where $\{y_1, \ldots, y_k\}$ is the range of attribute f. If p is the range from the declaration of f then $: \{X : p(X)\}$ is omitted. Two other statements $obs(l)$, where l is a literal, and $do(f(\bar{x}) = y)$, which we refer to as *activity records*, are used to record an observation of l being true and a deliberate intervention into a random experiment which assigned value y to $f(\bar{x})$. The rules of a P-log program Π define the collection of possible worlds – answer sets of the translation $\tau(\Pi)$ of Π into an ASP program. The sorts of the program are viewed as collections of atoms. For regular rules τ simply replaces P-log atoms of the form $f(\bar{x}) = y$ by $f(\bar{x}, y)$ and adds an axiom guaranteeing the uniqueness of y. The rule (1) is translated into

$$f(\bar{x}) = y_1 \ \mathbf{or} \ \ \ldots \ \mathbf{or} \ \ f(\bar{x}) = y_k \ \leftarrow body, \mathbf{not}\ intervene(f(\bar{x})) \tag{2}$$

(to avoid proliferation of notations we identify $f(\bar{x}) = y$ with $f(\bar{x}, y)$) and

$$intervene(f(\bar{x})) \leftarrow do(f(\bar{x}) = Y) \tag{3}$$

$$\leftarrow f(\bar{x}) = Y, \mathbf{not}\ p(Y), body, \mathbf{not}\ intervene(f(\bar{x})) \tag{4}$$

We also need

$$\leftarrow obs(l), \mathbf{not}\ l \tag{5}$$

$$f(\bar{x}) = y \leftarrow do(f(\bar{x}) = y). \tag{6}$$

Consider program G consisting of definition of $tests$, $grades$, $grade$ given above and rule $random(grade)$. It is easy to check that W_1 consisting of sort atoms $\{tests(t_1), tests(t_2), grades(a), \ldots, grades(d), grades(f)\}$ and a particular outcome $\{grade(t_1) = a, grade(t_2) = c\}$ of the grade assigning experiment is a possible world of G. Other possible worlds contain other assignments of grades to tests. To define the probabilistic semantics of a P-log program Π we need to define (unnormalized) probabilistic measure, $\mu(W)$ of possible worlds of Π quantifying the agent's belief in the likelihood of random experiments represented by W. The world W_1 above contains two such experiments.

In the absence of additional information P-log semantics uses the assumption (known as the Indifference Principle) that all possible outcomes of our experiments are equally likely. Therefore, $\mu(W_1)$ will be equal to $1/5 \times 1/5 = 1/25$. An

additional probabilistic information can be encoded in P-log by so called *causal probability statement* (or *pr*-atoms) – expressions of the form

$$pr(f(\bar{t}) = y|_c B) = v$$

where B is a set of literals possibly preceded by a default negation. The statement says that, if B holds, then the probability of the selection of y for the value of $f(\bar{t})$ is v. In addition, it indicates the potential existence of a direct causal link between B and the possible value of f. If we expand our program G by $pr(grade(T) = c) = 2/5$ then $\mu(W_1)$ would be equal to $2/5 \times 3/20$. As usual, the probability of a proposition is defined as the sum of normalized probabilistic measures of possible worlds in which this proposition is true.

Even though the authors believe that P-log is a very promising knowledge representation language which deserves further study and development, a critical reading of the original papers revealed that some of the design decisions made by the original authors can be substantially improved. This paper explains and justifies these improvements, makes necessary modifications of corresponding mathematical definitions, and demonstrates how the changes increase expressive power of the language.

2 Language Refinements

The first group of improvements, referred to as *language refinements*, clarifies syntax and semantics of the language and brings informal reading of its main concepts closer to their formal semantics.

2.1 Total and partial functions

The first clarification is related to the failure of the original definition of P-log to distinguish between *total* and *partial* functions. Even though the latter are clearly used in some examples, their existence is never explicitly mentioned in the text. Two things are needed to remedy the problem.

- An atomic statement of the form $f(\bar{x}) \neq y$ should be viewed as true when $f(\bar{x})$ is defined, i.e. has a value, and this value is different from y; A statement **not** $f(\bar{x}) = y$ should be true if $f(\bar{x})$ has the value different from y or is undefined.
- Literals of the form $f(\bar{x}) \neq y$ shall not be allowed in the heads of program rules.

The first requirement is rather obvious and, as was shown in [4] it can be achieved by a very small modification of ASP. To see the reason for the second requirement, let us consider the following:

Example 1. Let P_1 be a P-log program:

$$f : boolean$$
$$f \neq false$$

According to the intuitive reading of the program statement $f \neq false$ implies that f is defined and its value is different from $false$. Together with the declaration this should guarantee that the value of f is $true$. However, according to the original P-log semantics from [6], the program P_1 has exactly one possible world which consists of a literal $f \neq false$, and hence P_1 does not entail $f = true$.

The example shows an unpleasant discrepancy between intuitive meaning of the program and its formal semantics. Note, that the problem would disappear if f were defined as random. This would guarantee that f is defined and that it takes on one of the Boolean values. Hence, as expected, the new program, P_1' would entail $f = true$. The same effect, however, could be reached without allowing literals of the form $f(\bar{x}) \neq y$ in the heads of rules. In our case, $f \neq false$ of P_1' will be simply replaced by a more appropriate observation $obs(f \neq false)$ which will produce the same result. So, prohibiting occurrences of such literals in the heads allow us to narrow the gap between intuitive and formal meaning of statements without any real loss of expressive power. Moreover, disallowing this syntactic feature leads to a substantial simplification of the formal semantics of P-log. Instead of defining possible worlds as sets of literals we can view them simply as (partial) interpretations of the attribute terms from the program's signature (in other words, collections of atoms). So far we were not able to find any adverse effect of our restriction on the original syntax. It is important to notice that, despite this restriction, P-log programs may contain a rule $\neg p \leftarrow body$ where p is boolean. It is because $\neg p$ is understood simply as a shorthand for an atom $p = false$.

2.2 P-log observations

Another problem with the original P-log semantics is related to the intuitive meaning of P-log observations – statements of the form $obs(l)$. According to [6], such observations are used "to record the outcomes of random events, i.e., random attributes, and attributes dependent on them". However, axiom (5) does not faithfully reflect this intuition, since it does not prohibit observations of non-random events. Instead it simply views $obs(f(\bar{x}) = y)$ as a shorthand for the constraint

$$\leftarrow not\ f(\bar{x}) = y$$

where f is an arbitrary attribute. A newly introduced observation simply eliminates some of the possible worlds of the program, which reflects understanding of observations in classical probability theory. This view is also compatible with treatment of observations in action languages. So there are no adverse consequences of expanding the observability of attribute values to a non-random case. In fact, this is frequently done in practice. (See, for instance, "Bayesian squirrel" example in [6].) Hence, the new intuitive meaning allows observations of both, random and non-random events.

2.3 Refining the meaning of *do* statement

We also need to clarify the P-log meaning of the *do* statement. The original paper states: "the statement $do(f(\bar{x}) = y)$ indicates that $f(\bar{x}) = y$ is made true as a result of a deliberate (non-random) action". Note that here, $f(\bar{x})$ is not required to be declared as random, i.e., its value does not have to be normally defined by a random experiment. This is not wrong. Even though the original intervening action *do* of Pearl only applies to random attributes (no other types are available in Bayesian Nets) nothing prohibits us from expanding the domain of *do* to non-random ones. After all this is exactly what we did with observations. But, in case of intervening actions such extension seems to be unwarranted. It is easy to see that for non-random $f(\bar{x})$, $do(f(\bar{x}) = y)$ is (modulo *do*) equivalent to $f(\bar{x}) = y$, which undermines the utility of such statements. In addition, it violates one of the important principles of language design frequently advocated by N. Wirth and others: *Whenever possible, make sure that important type of informal statements you want expressible in your formal language corresponds to* one *language construct*. Moreover applying *do* to interfere into a random experiment with a dynamic range causes an ambiguity of an interpretation: should the deliberately assigned value belong to the dynamic range or an arbitrary value of the proper sort must be allowed? The formal semantics from [6] corresponds to the second option, but this seems to be accidental. It seems that the intuition and analysis of examples suggest that both, random and deliberately assigned values, should belong to the dynamic range of the attribute. This change can be achieved by a simple modification of translation τ of P-log into ASP.

We expand the signature of $\tau(\Pi)$ by a new atom $random(f(\bar{x}), p)$, identified with $random(f(\bar{x}) : \{X : p(X)\})$, replace rule (1) by

$$random(f(\bar{x}), p) \leftarrow body \tag{7}$$

and add constraints

$$\leftarrow not\ random(f(\bar{X}), p), do(f(\bar{X}) = Y) \tag{8}$$

and

$$\leftarrow f(\bar{x}) = Y,\ \mathbf{not}\ \ p(Y), random(f(\bar{x}), p) \tag{9}$$

Constraints guarantee that *do* can only be applied to random attributes and that both, random and deliberately assigned values are limited to those from the dynamic range. To describe results of random experiments which proceed without an intervention we introduce another special atom, $truly\_random(f(\bar{x}), p)$ which is true iff the value of $f(\bar{x})$ is assigned as the result of random experiment, and expand the program by rule

$$f(\bar{x}) = y_1\ \mathbf{or}\ \ \ldots\ \mathbf{or}\ \ f(\bar{x}) = y_k\ \ \leftarrow truly\_random(f(\bar{x}), p) \tag{10}$$

where $\{y_1, \ldots, y_k\}$ is the range of f, providing the means for random selection of the value for $f(\bar{x})$ and rule

$$\begin{array}{l} truly\_random(f(\bar{x})) \leftarrow random(f(\bar{x}), p), \\ \qquad\qquad\qquad\quad not\ do(f(\bar{x}) = y_1), \\ \qquad\qquad\qquad\quad \ldots, \\ \qquad\qquad\qquad\quad not\ do(f(\bar{x}) = y_k) \end{array} \tag{11}$$

which serves as the definition of *truly_random*. Even though the introduction of $truly\_random(f(\bar{x}), p)$ is not absolutely necessary (the body of rule (10) could have been replaced by $random(f(\bar{x}), p)$) it is better aligned with the English meaning of *random* and *truly_random* and slightly simplifies the assignment of probabilistic measures to possible worlds of Π. In particular, probabilistic measure should be assigned to $f(\bar{x})$ within a possible world W only if W contains $truly\_random(f(\bar{x}), p)$ for some p.

The differences between the behavior of the original P-log and its new incarnation can be illustrated by the following example:

Example 2. Consider P-log program P_2 consisting of rules

$$\begin{array}{l} f\ :\ \{1, 2, 3\} \\ random(f : \{X : X > 1\}) \\ do(f = 1) \end{array}$$

It is easy to see that, due to constraint (9), the program is logically inconsistent, i.e., has no possible worlds. According to the old semantics, however, P_2 has possible world $\{f = 1\}$. Similarly, constraint (8) ensures that program P_3

$$\begin{array}{l} f\ :\ \{1, 2, 3\} \\ do(f = 1) \end{array}$$

has no possible world while according to the old semantics its possible world is $\{f = 1\}$.

3 Language Enhancements

The proposed version of P-log contains two new enhancements:

- An extension of the language with a more elaborate and convenient type system similar to the one used in knowledge representation language SPARC [3].
- Removal of some restrictions on the occurrence of so called *activity records*, i.e. specialized literals formed by *obs* and *do*.

3.1 Sorts

Recall that, in the original P-log, sorts are described by statements of the form $s = \{t_1, \ldots, t_n\}$, where s is a sort name and $t_1, \ldots, t_n$ are ground terms. Even though this is sufficient theoretically, such an approach is almost impossible to use in practice. This is immediately obvious when we deal with large sorts which may possibly include each other. In the paper we extend the syntax for defining sorts by using the framework from [3] in which standard ASP syntax and semantics is expanded by a powerful type system. This supports the methodology of starting a knowledge representation task with identifying types of objects relevant to the domain, allows a much easier specification of complex and/or large sorts, helps to avoid potential problems with safety conditions of rules, and allows detection of trivial but difficult to detect errors related to typos and sort violations.

Sorts there are defined by statements of the form:

$$sort\_name = sort\_expression$$

where *sort_name* is a unique identifier preceded by the symbol # and *sort_expression* can be in one of the following forms:[1]

- $\{t_1, ..., t_n\}$
- $f(sort\_name_1, \ldots, sort\_name_n)$
- $sort\_name_1 \oplus \ldots \oplus sort\_name_n$

where $n > 0$, each t_i is a ground term, f is a function symbol, and $\oplus$ denotes a set operator (union, intersection, difference, or concatenation) where the set operations result in a non-empty set. This allows declarations of the form

```
#blocks = {b}{1..100}
#locations = #blocks + {table}
#actions = put(#blocks,#locations)
```

used for formalization of the blocks world domain. The first sort declaration concatenates b with the set of natural numbers from 1 to 100 obtaining blocks $\{b1, \ldots, b100\}$. The second uses union denoted by + to define locations as the set $\{b1, \ldots, b100, table\}$. The third defines the collection of actions *put* over the Cartesian product of the first two: $put(b1, b1), put(b1, b2), \ldots put(b1, table), \ldots$ Sorts of the latter form are sometimes referred to as *records*. The full language allows a more general form of record definition,

- $f(sort\_name_1(X_1), \ldots, sort\_name_n(X_n)) \ : \ cond$

where *cond* is a condition on variables $X_1, \ldots, X_n$. (For the precise form of this condition see [3].) For instance, the sort definition:

```
#actions = put(#blocks(X),#locations(Y)) : X <> Y)
```

[1] The actual syntax is slightly different, we simplify it here to shorten the description.

allows to avoid records like $put(b1, b1)$ which do not really denote any actions to be excluded from the sort.

We hope that this description is sufficient for most readers of this preliminary report. Those interested in more details are referred to [3] where this type system is discussed in depth.

3.2 Removing restriction on program occurrences of special atoms

Recall that, in addition to regular arithmetic and attribute atoms of the form $f(\bar{x}) = y$, the signature of P-log also contains atoms formed by four special Boolean attribute terms: $do(f(\bar{x}) = y)$, $obs(f(\bar{x}) = y)$, (often identified with $do(f(\bar{x}), y)$, $obs(f(\bar{x}), y)$) $obs(f(\bar{x}) \neq y)$ and $random(f(\bar{x}), p)$. Such atoms are also referred to as special. The original P-log, however, severely restrict occurrences of these atoms in the program rules. Atoms formed by *do* and *obs* – referred to as *activity records* – are only allowed to occur in a program as facts (rules with the empty bodies). Atoms formed by *random* are only allowed to occur in the head of rules. The new version of P-log removes this restriction. The original reason to do this was not caused by "practical" need of such generalized rules.

We followed language design principles illuminated by N. Wirth [18] and attempted to simplify P-log by removing restrictions, i.e., "achieving simplicity by generalization". This attempt seems to be successful. The syntax of the language was simplified. The semantics remained unchanged. Addition of new concepts provided us with the ability to express new, previously inexpressible, thoughts. However, as pointed out in [18] one should remember that achieving simplicity through generality has its possible drawbacks. In doing this we "may end up with programs that through their very conciseness and lack of redundancy elude our limited intellectual ability". So far, we were not able to find any serious adverse effects caused by our generalization, but one should remember that this report is preliminary.

We now discuss a particularly interesting case of our generalization: rules with activity records in their bodies. We will show how such rules allow us to express knowledge we were not able to express in the original P-log (where activity records were only allowed to occur as facts). We have already mentioned that the addition of an observation to an original P-log program may only eliminate some of its possible worlds but cannot create a new one. Allowing observations to occur in the bodies of P-log rules changes the situation. The addition of an observation $obs(a, true)$ to a program

$$Q = \begin{cases} a : \ boolean \\ \neg a \leftarrow not\ a \\ a \leftarrow obs(a, true) \end{cases}$$

creates a possible world which did not exist according to the original program. This extension of the language does not significantly complicate its mathematical semantics but seems to add substantially to its expressive power.

To further illustrate this phenomena let us assume that we would like to use P-log to formalize knowledge relevant to the following problem.

Example 3. Suppose that an experienced diagnostician was able to reduce an appearance of a malfunctioning symptom s to two possible causes, c_1 and c_2. In what follows we discuss several ways in which this knowledge can be represented and used in reasoning.

The purely qualitative information available to the diagnostician can be expressed in P-log by a program P_4:

$$P_4 = \begin{cases} s, c_1, c_2 : \ \ boolean \\ \quad \neg c_1 \leftarrow not\ c_1 \\ \quad \neg c_2 \leftarrow not\ c_2 \\ \quad\quad s \leftarrow c_1 \\ \quad\quad s \leftarrow c_2 \\ \quad \neg s \leftarrow not\ s \end{cases}$$

The first two rules say that malfunctioning does not normally happen – a natural default we use in our actions before becoming aware of a problem by observing or experiencing its symptoms. The next three rules give the complete list of possible causes for s. According to this program the probabilities of s, c_1 and c_2 are 0. If P_4 were expanded by, say, c_1, the new program $P_4 \cup \{c_1\}$ would entail the symptom s. Note also, that under no circumstances the diagnostician is expected to expand P_4 by the fact s. That would amount to accepting s as its own cause and making connections between c_1 and c_2 unusable for diagnostics. Instead, according to a standard methodology, a diagnostician who observes the symptom is expected to record it by statement $obs(s, true)$. In our case this, however, leads to a problem. It is important to note that an update of P_4 by observation of the truth of any attribute of P_4 (including s) leads to inconsistency. This is not necessarily an unwelcome outcome for the observations of causes – after all causes are normally not directly observable and need to be derived from the observations of symptoms and the background knowledge. This shall not however happen for the observation of the symptom s. The following informal argument is possible in this case: *Since we are given a complete list of possible causes of s and s is observed to be true we can not continue to use closed world assumptions for causes. At least one of them must be true to explain s. But which one and with what probability? To answer this question we should think of causes as random attributes which may or may not be true.* Accordingly, the program describing the agent's knowledge should have possible worlds $W_1 = \{c_1, s\}$, $W_2 = \{c_2, s\}$, and $W_3 = \{c_1, c_2, s\}$ corresponding to three combinations of possible causes of s.

The missing knowledge used by the reasoner to go from observation $obs(s)$ of a symptom to its causes can be represented by expanding P_4 by the rules:

$$R = \begin{cases} random(c_1) \leftarrow obs(s) \\ random(c_2) \leftarrow obs(s) \end{cases}$$

which have observations in their bodies. It is easy to check that program

$$P_5 = P_4 \cup R \cup \{obs(s)\}$$

is consistent and has three possible worlds W_1, W_2, and W_3 described above. Using the Indifference Principle built into the semantics of P-log, the program assigns probability 2/3 to both c_1 and c_2 and probability 1 to s. Note that having $obs(s)$ instead of s in the bodies of rules from R is important. Using s would lead to some unexpected behaviour. To see that, consider our original knowledge base (formalized by $T = P_4 \cup R$) and assume that diagnostician discovered (or assumed) that c_1 is true. Of course, that will allow the diagnostician to derive s. The new information, however, will not change the probability of c_2. It will remain 0. This is exactly the result produced by $T \cup \{c_1\}$. Let us now see what happens if $obs(s)$ in R is replaced by s. Let us denote the result of this replacement by R' and consider

$$Q = P_4 \cup R' \cup \{c_1\}$$

Clearly, Q entails $random(c_2)$ and, hence, the addition of c_1 causes the probability of c_2 change from 0 to some positive value, which violates our intuition.

Let us now assume that, by checking some available statistics, the diagnostician acquires knowledge about probabilities of c_1 and c_2. These probabilities can be represented by the set PA of causal probability atoms:

$$PA = \begin{cases} pr(c_1) = 0.05 \\ pr(c_2) = 0.01 \end{cases}$$

The probabilities assigned to c_1 and c_2 by the new program,

$$P_6 = P_5 \cup PA$$

are now approximately 0.8 and 0.2. So c_1 is the most likely cause of the symptom.

Finally, let us consider the case when after some direct or indirect observation the diagnostician establishes that c_2 is true. The probabilities assigned to c_1 and c_2 by program

$$P_7 = P_6 \cup \{obs(c_2)\}$$

are now 0.05 and 1 respectively. The latter observation is an example of a probabilistic phenomena called "explaining away" [17]: when you have competing possible causes for some event, and the chances of one of those causes increases, the chances of the other causes must decline since they are being "explained away" by the first explanation.

The example shows a fairly seamless combination of logical and probabilistic reasoning in search of causal explanations of a symptom. The authors were not able to express this type of reasoning in the original P-log based on ASP. We could, however, do it in CR-Prolog – extension of ASP by so called consistency-restoring rules. This, however, required to learn the semantics of CR-Prolog.

Moreover, currently there is no reasoning system implementing P-log with consistency-restoring rules and the task of developing and efficiently implementing such a system seems to be non-trivial. In contrast, implementing P-log with rules containing activity records in their bodies seem to be substantially less formidable.

There are other possible uses of observations in the body of rules. Not all attributes are always observable. In fact, in the original interpretation of *obs*, if the value of $f(\bar{x})$ is undefined, then $f(\bar{x})$ is, of course, unobservable. Sometimes, however, $f(\bar{x})$ is undefined by a program not because such value does not exist, but simply because it is not known to the reasoner. In some of such cases this value can be obtained by a direct observation. (We refer to such $f(\bar{x})$ as *directly observable*.) In P-log this property can be expressed by a rule:

$$f(\bar{x}) = y \leftarrow obs(f(\bar{x}), y)$$

Note that, for a directly observable value y of an attribute $f(\bar{x})$, expanding a program by $obs(f(\bar{x}), y)$ is (modulo atoms formed by *obs*) equivalent to expanding it by $f(\bar{x}) = y$. Impossibility of observing $f(\bar{x})$ can be expressed as

$$\leftarrow obs(f(\bar{x}), Y)$$

4 Conclusion

The paper is a preliminary report on a new, modified version of knowledge representation language P-log. The changes are not dramatic and may be even missed by a casual observer, but we believe them to be important. In particular, we

- Increase integrity of the language by correcting omissions and inaccuracies of the original version and bringing formal and informal meanings of important constructs closer to each other.
- Supply P-log with a type system which supports methodology of starting a knowledge representation task with identifying and declaring types of objects relevant to the domain.
- Simplify P-log by generalizing its syntax. The old restrictions on the occurrence of special atoms in programs rules are removed and treated as regular ones. In addition to achieving simplicity, the new version gives knowledge engineers means to express knowledge and automate reasoning which were not available in the language before. As an example, we discuss how the new version can account for the common probabilistic phenomenon of "explaining away".

As a result, the new version should be significantly better suited for teaching as well as for practical applications. Another extension of the original P-log, strongly advocated by the second author, is that by aggregates and other set related constructs of Alog [11, 12]. Such an extension, currently discussed by the

authors, will not require any additional theoretical work even though it will have some influence on the language implementation.

Future work

In the near future, we plan to:

- Implement a query answering system for the new version of P-log.
- Further investigate the expressive power and usability of new, generalized rules, and type system.
- Consider further extensions of the language.

Currently, we are working on the development of a new P-log query answering system[2] which improves the algorithms from [20] (Short description of the corresponding algorithm can be found in [1]. Our improvements include: expanding the class of programs where the algorithms are applicable, implementing a type system, and softening some of the restrictions on the programs. We plan to finish this implementation and its correctness proof in the near future. It would also be interesting to investigate other approaches to P-log inference, including:

- Developing approximate inference algorithms (possibly based on Monte-Carlo sampling methods).
- Comparing inference methods for a recently introduced new formalism LP^{MLN} [13] with those in P-log developed so far. The results from [14] and [2] would allow us to use inference in P-log for programs in LP^{MLN}, and vice versa.

Acknowledgments

We would like to thank Leroy Mason and Nelson Rushton for useful discussions on subjects related to this paper.

References

1. Balai, E.: Combining logic and probability: P-log perspective. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016. pp. 3974–3975 (2016)
2. Balai, E., Gelfond, M.: On the relationship between P-log and LP^{MLN}. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016. pp. 915–921 (2016)
3. Balai, E., Gelfond, M., Zhang, Y.: Towards answer set programming with sorts. In: Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013, Corunna, Spain, September 15-19, 2013. Proceedings. pp. 135–147 (2013)

[2] https://github.com/iensen/plog2.0/wiki

4. Balduccini, M.: A "conservative" approach to extending answer set programming with non-Herbrand functions. In: Correct Reasoning, pp. 24–39. Springer (2012)
5. Balduccini, M., Gelfond, M.: Logic programs with consistency-restoring rules. In: International Symposium on Logical Formalization of Commonsense Reasoning, AAAI 2003 Spring Symposium Series. pp. 9–18 (2003)
6. Baral, C., Gelfond, M., Rushton, J.N.: Probabilistic reasoning with answer sets. TPLP 9(1), 57–144 (2009)
7. Baral, C., Gelfond, M., Rushton, N.: Probabilistic reasoning with answer sets. In: International Conference on Logic Programming and Nonmonotonic Reasoning. pp. 21–33. Springer (2004)
8. Baral, C., Hunsaker, M.: Using the probabilistic logic programming language p-log for causal and counterfactual reasoning and non-naive conditioning. In: IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007. pp. 243–249 (2007)
9. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing 9(3/4), 365–386 (1991)
10. Gelfond, M., Rushton, N.: Causal and probabilistic reasoning in P-log. In: Dechter, R., Geffner, H., Halpern, J. (eds.) A tribute to Judea Pearl, pp. 337–359. College Publications (2010)
11. Gelfond, M., Zhang, Y.: Vicious circle principle and logic programs with aggregates. TPLP 14(4-5), 587–601 (2014), http://dx.doi.org/10.1017/S1471068414000222
12. Gelfond, M., Zhang, Y.: Vicious circle principle and formation of sets in ASP based languages. In: Logic Programming and Nonmonotonic Reasoning, 14th International Conference, LPNMR 2017 (to appear)
13. Lee, J., Wang, Y.: Weighted rules under the stable model semantics. In: Baral, C., Delgrande, J.P., Wolter, F. (eds.) Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016. pp. 145–154. AAAI Press (2016), http://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12901
14. Lee, J., Yang, Z.: LP^{MLN}, weak constraints, and p-log. In: Singh, S.P., Markovitch, S. (eds.) Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA. pp. 1170–1177. AAAI Press (2017), http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14547
15. Pearl, J.: Causality: Models, Reasoning, and Inference. Cambridge University Press (2000)
16. Pereira, L.M., Saptawijaya, A.: Programming Machine Ethics. Springer Publishing Company, Incorporated, 1st edn. (2016)
17. Wellman, M.P., Henrion, M.: Explaining'explaining away'. IEEE Transactions on Pattern Analysis and Machine Intelligence 15(3), 287–292 (1993)
18. Wirth, N.: On the design of programming languages. In: IFIP Congress. pp. 386–393 (1974)
19. Zhang, S., Stone, P.: Corpp: Commonsense reasoning and probabilistic planning, as applied to dialog with a mobile robot. In: Proceedings of the 29th Conference on Artificial Intelligence (AAAI) (January 2015)
20. Zhu, W.: Plog: Its algorithms and applications. Ph.D. thesis, Texas Tech University (2012)