Combining Logical and Probabilistic Reasoning

Michael Gelfond and **Nelson Rushton** and **Weijun Zhu**
Computer Science Department
Texas Tech University
Lubbock, TX 79409 USA
{mgelfond,nrushton,weijun.zhu}@cs.ttu.edu

Abstract

This paper describes a family of knowledge representation problems, whose intuitive solutions require reasoning about defaults, the effects of actions, and quantitative probabilities. We describe an extension of the probabilistic logic language P-log (Baral & Gelfond & Rushton 2004), which uses "consistency restoring rules" to tackle the problems described. We also report the results of a preliminary investigation into the efficiency of our P-log implementation, as compared with ACE(Chavira & Darwiche & Jaeger 2004), a system developed by Automated Reasoning Group at UCLA.

Introduction

P-log, introduced in (Baral & Gelfond & Rushton 2004), is a language for representing and reasoning with logical and probabilistic knowledge. P-log is based on Answer Set Prolog(Baral 2003)(Gelfond & Lifschitz 1988), and inherits all of its expressive power, allowing us to represent recursive definitions, and both classical and default negation. It also allows us to represent and reason with quantitative probabilities, building on the theory of causal Bayesian networks (Pearl 2000), and combining them with the logical mechanisms of Answer Set Prolog.

Moreover, P-log has a powerful mechanism for *updating*, or reassessing probabilities in the light of new information. This mechanism allows to represent and reason with updates which, in the framework of classical probability, could not be expressed using conditioning and would require the construction of a new model. For example, classical conditioning can only handle updates which eliminate possible worlds (aka outcomes) from the model, and such that set of outcomes remaining has nonzero prior probability. P-log, in contrast, can handle updates which introduce new possible worlds or change the probabilities of existing worlds, as well as some updates which have zero prior probability. This paper introduces a modification of P-log which allows an even wider variety of updates, by adding the use of "consistency-restoring rules", as defined in section on our language.

Besides expressive power, there is another reason to combine logical and probabilistic reasoning. Recently, research in Bayesian networks has shown that logical relationships between variables in a network can be used to enhance the efficiency of computations done using the network — essentially by eliminating impossible combinations of the values of variables during processing. Our implementation of P-log is based on Smodels (Niemelä & Simons 1997), which can find the possible worlds of a logic program quickly. This raises the question of how our P-log implementation can perform, on similar problems, side by side with other systems based on Bayes nets. Though our current P-log implementation, available at www.krlab.cs.ttu.edu/software, is only a prototype and is not "built for speed", our preliminary investigations into its efficiency look promising. The final sections of this paper compare our P-log prototype with ACE (Chavira & Darwiche & Jaeger 2004), a state-of-the-art system for computing with Bayes nets with logical information, on a handful of benchmark problems taken from (Chavira & Darwiche & Jaeger 2004). The experiment described here is too small to support conclusions about the relative speed of the two systems; but we feel the results obtained so far are worthy of interest.

This paper is arranged as follows. First, we first describe the "robot problem", a problem in knowledge representation. Next, we briefly describes the language we will use to represent and reason about this problem. We then describe how the robot problem can be represented and reasoned with in our language. Finally, we give the results of an experiment on performance, and a note comparing our representation with that of relational Bayes nets (Jaeger 2004) .

A Moving Robot

We consider a formalization of a problem whose original version, not containing probabilistic reasoning, first appeared in (Iwan & Lakemeyer 2002). A P-log representation was presented in (Baral & Gelfond & Rushton 2004). This paper will present new extensions on the problem, and demonstrate how they can be approached using P-log with consistency restoring rules.

Part 1. There are rooms, say r_0, r_1, r_2, reachable from the current position of a robot. The rooms can be open or closed. The robot cannot open the doors. It is known that the robot navigation is usually successful. However, a (rare) malfunction can cause the robot to go off course and enter any one of the open rooms.

We want to be able to use the corresponding formalization for correctly answering simple questions about the robot's behavior. Here are some typical scenarios:

1. At time 0 the robot moved toward open room r_1. Where will the robot be after the operation is completed? The expected answer, based on the default "robot navigation is usually successful", is *in room* r_1.

2. Initially doors to all three rooms are open. The robot moved toward open room r_1 but found itself in some other room. Why? The only reasonable conclusion we can obtain from our knowledge base is *the robot malfunctioned*. Where is the robot now? The expected answer is *in room* r_0 *or* r_2.

3. Initially r_0 and r_1 are open, and r_2 is closed. The rest is as in scenario 2. Now the answer to "where is the robot now" is *in room* r_0.

Part 2. Let us now expand the example by some probabilistic information. Assume for instance that the probability of a malfunctioning robot going to the intended room is 0.5. If it does not enter the intended room, it will enter another room at random. We would like to consider the following scenarios:

1. Initially doors to all three rooms are open. The robot moved toward room r_1. What possible locations can the robot have now? The intuitive answer, based on the default that robot navigation is usually successful, is that the robot is in r_1.

2. Initially doors to all three rooms are open. The robot moved toward open room r_1 but found itself in some other room. What possible locations can the robot have now, and what are their respective probabilities? The intuitive answer is that the robot could be in either r_0 or r_2, each with probability $1/2$. This answer requires the calculation of numerical probabilities for each of the possible resulting situations.

Even though the example is simple, the agent capable of obtaining the above conclusions should be able to reason with defaults and their exceptions, and make conclusions about effects of (sometimes non-deterministic) actions. Its reasoning will be non-monotonic, i.e. capable of revising its conclusions as the result of new information. The agent must also be capable of reasoning with quantitative probabilistic information.

The Language

CR-Prolog

We start with a description of syntax and semantics of a subset of the logic-programming language CR-Prolog, suitable for our purpose.

A program of CR-Prolog is pair consisting of signature and a collection of rules of the form:

$$l_0 \leftarrow l_1, \ldots, l_n \tag{1}$$

and

$$r : l_0 \text{ +- } l_1, \ldots, l_n \tag{2}$$

where $l_1, \ldots, l_n$ are literals, and r is a term representing the name of the rule it precedes. Rules of type (1) are called *regular*; those of type (2) are called *consistency restoring* rules (cr-rules). The set of regular rules of a cr-program Π will be denoted by Π^r; the set of cr-rules of Π will be denoted by Π^{cr}.

Definitions:
A regular rule $\alpha(r)$ obtained from a consistency restoring rule r by replacing +- by $\leftarrow$, and omitting the name of the rule, will be called the *regular counterpart* of r.

As usual, semantics is given with respect to ground programs, and a rule with variables is viewed as shorthand for schema of ground rules. A minimal (with respect to set theoretic inclusion) collection R of cr-rules of Π such that $\Pi^r \cup \alpha(R)$ is consistent (i.e. has an answer set) is called an *abductive support* of Π.

A set A is called an answer set of Π if it is an answer set of $\Pi^r \cup \alpha(R)$ for some abductive support of Π.

A program Π is called *categorical* if it has exactly one abductive support.

Examples.
Consider a program

```
p(X)  :- not ab(X).
ab(e1).
ab(e2).
q(e).
r(X) :- p(X),q(X).
ab(X) +-.
```

The program includes a default with two exceptions, a partial definition of r in terms of p and q, and consistency restoring rule which acknowledges the possible existence of unknown exceptions to the default. Since such a possibility is ignored whenever consistent results can be obtained without considering it, the unique answer set of the above program is $\{ab(e1), ab(e2), q(e), p(e), r(e)\}$.

Suppose now that the program is expanded by a new atom, -$r(e)$. The regular part of the new program has no answer set. The cr-rule solves the problem by assuming that e is a previously unknown exception to the default. The resulting answer set is $\{ab(e1), ab(e2), q(e), ab(e)\}$.

P-log

A complete description of P-log can be found in our paper, currently under consideration by *Theory and Practice of Logic Programming*, available at http://www.krlab.cs.ttu.edu/Papers/download/bgr05.pdf.
Here, we describe a fragment of the language sufficient for current purposes.

The signature of a P-log program 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*. An attributes a with domain d and range r is declared in a P-log program as

$$a : d \rightarrow r \tag{3}$$

Technically, every P-log atom is of the form $a(\bar{t}) = y$, where a is an attribute, $\bar{t}$ is a vector of terms, and y is a term. However, if an attribute a is declared as a Boolean attribute (i.e., an attribute with range $\{true, false\}$), then we may write $a(\bar{t})$ and $-a(\bar{t})$ as shorthand for the atoms $a(\bar{t}) = true$ and $a(\bar{t}) = false$, respectively.

Besides declarations, a P-log program consists of two parts: its logical part and its probabilistic part. The logical part essentially consists of a logic program together with a collection of *random selection rules*, which take the form

$$[\ r\ ]\ random(a(\bar{t}) : \{X : p(X)\}) \leftarrow B. \tag{4}$$

Here, r is a term used to name the rule and B is a collection of extended literals (where an *extended literal* is either an atom or an atom preceded by not). Names are optional and can be omitted if the program contains exactly one random selection for $a(\bar{t})$. The above random selection rule roughly says that *if B holds, the value of $a(\bar{t})$ is selected at random from $\{X : p(X)\} \cap range(a)$*. If B is empty we simply write

$$[\ r\ ]\ random(a(\bar{t}) : \{X : p(X)\}). \tag{5}$$

In case the value of $a(\bar{t})$ is selected at random from its entire range (which is quite often the case), we write

$$[\ r\ ]\ random(a(\bar{t})) \leftarrow B. \tag{6}$$

We obtain the possible worlds of a P-log program Π by translating its logical part into an Answer Set Prolog program $\tau(\Pi)$ and identifying answer sets of $\tau(\Pi)$ with possible worlds of Π. An atom of the form $a(\bar{t}) = y$ is translated as an atom $a(\bar{t}, y)$. For each attribute a of Π, $\tau(\Pi)$ contains the axiom

$$\neg a(\bar{t}, Y_1) \leftarrow a(\bar{t}, Y_2), Y_1 \neq Y_2 \tag{7}$$

which says that a defines a partial function. The random selection rule

$$[\ r\ ]\ random(a(\bar{t}) : \{X : p(X)\}) \leftarrow B. \tag{8}$$

is translated to the following choice rule in Answer Set Prolog

$$1\{a(\bar{t}, Z) : c_0(Z) : p(Z)\}1 \leftarrow B. \tag{9}$$

where c_0 is the range of a. The precise semantics of this rule can be found in (Simons 1999). It says roughly that every answer set s satisfying B must also satisfy $a(\bar{t}, Z)$ for some Z such that $c_0(Z) \in s$ and $p(Z) \in s$.

The *probabilistic part* of a P-log program consists of a collection of *probability atoms*, which have the form:

$$pr_r(a(\bar{t}) = y \mid_c B) = v \tag{10}$$

where $v \in [0, 1]$, B is a collections of extended literals, and r is the name of a random selection rule for $a(\bar{t})$. The above probability atom says that *if B were to be true, and the value of $a(\bar{t})$ were selected by rule r, then B would cause $a(\bar{t}) = y$ with probability v*.

If probability atoms are not given for all possible outcomes of a random selection, then all outcomes of the selection which are not assigned probabilities are considered to be equally likely by default. This semantic is explained in detail in our paper available at http://www.krlab.cs.ttu.edu/Papers/download/bgr05.pdf.

A *CR-P-log program* Π consists of P-log program Π_1, together with a (possibly empty) collection Π_2 of CR-rules, such that $\tau(\Pi_1) \cup \Pi_2$ is a categorical program. The possible worlds of Π are computed as in the previous section. A probability measure on these worlds is obtained in the usual way, using the probability atoms of Π.

Formalization

We start with formalizing the knowledge in Part 1 of our problem description. We need the initial and final moments of time, the rooms, and the actions.

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

We will need actions:

$go\_in\ :\ rooms \rightarrow boolean.$

$break\ :\ boolean$

The action $go\_in(R)$ consists of the robot *attempting* to enter a room R at time step 0. $break$ is an exogenous breaking action which may occur at moment 0 and alter the outcome of this attempt. In what follows, (possibly indexed) variables R will be used for rooms.

A state of the domain will be modeled by a time-dependent attribute, in, and a time independent attribute $open$.

$open : rooms \rightarrow boolean.$

$in : time \rightarrow rooms.$

The description of dynamic behavior of the system will be given by the rules below:

The first two rules state that the robot navigation is usually successful, and a malfunctioning robot constitutes an exception to this default.

1. $in(1) = R \leftarrow go\_in(R), \mathbf{not}\ ab.$
2. $ab \leftarrow break.$

The random selection rule (3) below plays a role of a (nondeterministic) causal law. It says that a malfunctioning robot can end up in any one of the open rooms.

3. $[r]random(in(1) : \{X : open(X)\}) \leftarrow go\_in(R), break.$

We also need inertia axioms for the fluent in.

4a. $in(1) = R \leftarrow in(0) = R, \mathbf{not}\ \neg in(1) = R.$
4b. $in(1) \neq R \leftarrow in(0) \neq R, \mathbf{not}\ in(1) = R.$

Finally, we assume that only closed doors will be specified in the initial situation. Otherwise doors are assumed to be open.

5. $open(R) \leftarrow \mathbf{not}\ \neg open(R).$

The resulting program, Π_0, completes the first stage of our formalization. The program will be used in conjunction with a collection X of atoms of the form $in(0) = R$, $\neg open(R)$,

$go\_in(R)$, $break$ which satisfies the following conditions: X contains at most one atom of the form $in(0) = R$ (robot cannot be in two rooms at the same time); X has at most one atom of the form $go\_in(R)$ (robot cannot move to more than one room); X does not contain a pair of atoms of the form $\neg open(R)$, $go\_in(R)$ (robot does not attempt to enter a closed room); and X does not contain a pair of atoms of the form $\neg open(R)$, $in(0) = R$ (robot cannot start in a closed room). A set X satisfying these properties will be normally referred to as a *valid input* of Π_0.

Given an input $X_1 = \{go\_in(r_0)\}$ the program $\Pi_0 \cup X_1$ will correctly conclude $in(1) = r_0$. The input $X_2 = \{go\_in(r_0), break\}$ will result in three possible worlds containing $in(1) = r_0, in(1) = r_1$ and $in(1) = r_2$ respectively. If, in addition, we are given $\neg open(r_2)$ the third possible world will disappear, etc.

Now let us expand Π_0 by some useful probabilistic information. We can for instance consider Π_1 obtained from Π_0 by adding the probability atom:

8. $pr_r(in(1) = R \mid_c go\_in(R), break) = 1/2$.

Program $T_1 = \Pi_1 \cup X_1$ has the unique possible world which contains $in(1) = r_0$.

Now consider $T_2 = \Pi_1 \cup X_2$. It 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. $P_{T_2}(W_0)$ is assigned a probability of $1/2$, while $P_{T_2}(W_1) = P_{T_2}(W_2) = 1/4$ by default. Therefore $P_{T_2}(in(1) = r_0) = 1/2$. Here the addition of $break$ to the knowledge base changed the degree of the reasoner's belief in $in(1) = r_0$ from 1 to $1/2$. This is not possible in classical Bayesian updating, for two reasons. First, the prior probability of $break$ is 0 and hence it cannot be conditioned upon. Second, the prior probability of $in(1) = r_0$ is 1 and hence cannot be diminished by classical conditioning. To account for this reasoning in the classical framework requires the creation of a new probabilistic model. However, each model is a function of the underlying knowledge base; and so P-log allows us to represent the change in the form of an update.

Our analysis so far has assumed that our agent is given knowledge of whether a break occurred. Suppose, however, that this knowledge may be left out; i.e., that $break$ may be absent from our input even if a break occurred. This intuitively allows inputs such as $\{go\_in(r_1), in(1) = r_2\}$, which, when added to program Π_1, result in inconsistency (in the sense of having no possible worlds). We can represent the belief that breaks are rare, but may occur without our being informed, by the following cr-rule:

9. $break$ +-

Let Π_2 consist of Π_1 together with this rule. $\Pi_2 \cup X_1$ has the same answer sets as $\Pi_1 \cup X_1$ — in this case our reasoning agent is not forced to consider the possibility of a break. Now let $X_3 = \{go\_in(r_0), in(1) \neq r_0\}$. $\Pi_1 \cup X_3$ is inconsistent — it has no answer sets. $\Pi_2 \cup X_3$, however, has two answer sets: one containing $in(1) = r_0$ and one containing $in(1) = r_2$. Each of these answer sets has probability $1/2$ according to the semantics of our language, which accords with intuition.

A note on performance

The random blocks problem was introduced by (Chavira & Darwiche & Jaeger 2004) to demonstrate the performance of ACE, a system for probabilistic inference based on relational Bayesian networks. 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 of a location for each block. A path from one location to another is defined in the natural way, and a path cannot go through a location with a block in it. A typical query might ask for the probability that there is an unblocked path between two locations L_1 and L_2, where the placement of the blocks is uncertain.

The following program describes a situation with five locations and two blocks. It contains two sorts, one for blocks and one for locations.

```
#domain location(L; L1; L2; L3).
#domain block(B; B1).
location={1..5}.  block={a, b}.
```

The arrangement of the locations is represented by the relations *left_of* and *below*.

```
left_of(1, 2). left_of(2, 3).
left_of(4, 5). below(4, 2).
below(5, 3).
```

The attribute *place* maps each block to a location.

```
place: block -> location.
```

Block a can be placed at any location.

```
[r1]  random(place(a)).
```

Block b can be placed at a location that is not occupied by block a. We introduce a unary predicate $free\_for\_b$, which is satisfied by all the candidate locations that block b can occupy. In this problem, $free\_for\_b(L)$ is true if block a is not at L.

```
[r2]  random(place(b):
         {X:free_for_b(X)}).

free_for_b(L):- not place(a,L).
```

Now we are going to define the connectivity of the map. First, we say a location L is blocked if there is a block at L.

```
blocked(L):- place(B) = L.
```

The following two rules claim that for any two locations L_1 and L_2, if they satisfy the *left_of* or $below$ relation, and there is no block placed at either of them, then L_1 and L_2 are connected. The logical operator not represents negation as failure. Its precise semantics are described in (Gelfond & Lifschitz 1988).

```
connected(L1,L2) :- left_of(L1,L2),
          not blocked(L1),
          not blocked(L2).
```

```
connected(L1,L2) :- below(L1,L2),
          not blocked(L1),
          not blocked(L2).
```

We also need the following rules to state that the relation $connected$ is symmetric and transitive. (Note $neq(X, Y)$ is a built-in relation meaning that X and Y are not equal.)

```
connected(L1,L2) :- connected(L2,L1),
                    neq(L1,L2).

connected(L1,L3) :- connected(L1,L2),
                    connected(L2,L3),
                    neq(L1,L3).
```

The last rule says if there is no reason to believe two locations L_1 and L_2 are connected, then they are not connected.

```
-connected(L1,L2) :-
         not connected(L1,L2).

%The end of the program.
```

We compare the performance of P-log with ACE (Chavira & Darwiche & Jaeger 2004), a system developed by Automated Reasoning Group at UCLA. Our test data sets are taken directly from (Chavira & Darwiche & Jaeger 2004). Each test set has a name of the form $blockmap - l - b$, where l is the number of locations, and b is the number of blocks in the example. The experiments are run for both reasoning systems on on a 1.8GHZ Pentium M with 1GB of RAM.

Problem	Ans	P-log	ACE
blockmap-05-01	5	0.119	1.06
blockmap-05-02	20	0.140	1.23
blockmap-05-03	60	0.209	1.442
blockmap-10-01	10	0.151	13.26
blockmap-10-02	90	0.489	22.88
blockmap-10-03	720	1.763	34.6
blockmap-15-01	15	0.380	55.91
blockmap-15-02	210	1.230	118.19
blockmap-15-03	2730	9.752	327.91
blockmap-20-01	20	0.781	301.41
blockmap-20-02	380	3.655	642.71
blockmap-20-03	6840	52.194	2737.6
blockmap-22-01	22	0.920	773.67
blockmap-22-02	462	4.956	1448.34
blockmap-22-03	9240	67.977	>1800

Table 1: P-log and ACE's performance on random blocks problem. Times are given in seconds.

The third column gives the compilation time in seconds of each problem in ACE. In ACE, once the model is compiled, queries can be answered relatively quickly, compared with the time to compile (around an order of magnitude less than compile time).

The first column gives the number of possible worlds of each problem in P-log. The second column gives the time in seconds for our P-log implementation to obtain the possible worlds of the given problem, along with their probabilities. This represents a lower bound on the time for P-log to answer queries about unconditional probabilities. As in ACE, the additional time for a particular query is relatively small. One can see that the run time of P-log is basically a function of the number of possible worlds considered. Our current prototype implementation does not store these possible worlds for later use, so they must be recomputed for each query. However, if our query involves probabilities conditioned on some set of observations, then possible worlds failing to satisfy these observations are never generated, which can decrease the time and space requirements dramatically.

P-log seems to perform well on this small data set. The random block problem combines uncertainty (in the placement of blocks) with a recursively defined logical relation ($connected$). The relation $connected$ is a symmetric and transitive relation, leading to a cycled dependencies, which are handled quite efficiently by Smodels (Niemelä & Simons 1997), the system on which our P-log solver rests. This may help explain the efficiency of P-log in this problem. Since ACE is a system fine tuned and well noted for its efficiency, we were frankly quite surprised to find that our naive prototype performed comparably or better with ACE on this sample problem. Note that we do not make any general claims about the comparative efficiency of the P-log and ACE, and this is a topic for further investigation.

Representation in P-log compared with Relational Bayesian Networks

The representation language of ACE is that of relational Bayes nets (RBN's) (Jaeger 2004). The objective of RBN's is to succinctly represent Bayes nets with variables: roughly speaking, RBN's are to Bayes nets as predicate logic is to propositional logic. The objective of P-log, on the other hand, is to represent logical and probabilistic beliefs of an agent. Differences in elegance and expressiveness of the languages stem largely from these differences in objective.

A Bayesian networks may exhibit determinism at some of its nodes, corresponding to the case where the probability of any value of a node, conditioned on its parents, is 1 or 0. This special case amounts to the node being a Boolean function of its parents, and so RBN's come with a natural, though slightly inelegant method for expressing Boolean logic. Among its other features, relational Bayes nets introduce primitives for making these representations less inelegant.

The logical expressiveness available in Primula, or any system whose underlying representation is a Bayes net, naturally consists of those constructs which crystallize out of local determinism in Bayesian networks. On the other hand, the logical apparatus of P-log, Answer Set Prolog, is the result of deliberate and long research in non-monotonic logic – and so we should expect some advantages in elegance and/or expressiveness with respect to complex logical elements of a problem. For example, we do not see how to easily represent the robot example of this paper as a relational Bayes network — as it contains defaults, closed world assumptions, inertia

axioms, and can be queried with evidence consisting of interventions using Pearl's do-calculus.

For comparison we include some of the source code used by ACE for the random blocks problem. The relation *connected* is defined in the above P-log program by the four rules containing *connected* in their left hand sides, plus a closed world assumption. These four rules straightforwardly say that two blocks are connected if one is below or to the left of the other and neither is blocked, and that the relation is symmetric and transitive. These P-log rules are comparable in their content to the following code in the language of RBN's. The precise semantics of the code can be found in the Primula user's guide, which can be downloaded at http://www.cs.aau.dk/ jaeger/Primula/more.htm.

```
% The following definitions describe
% when two blocks are connected

connected(l1,l2)=
  (sformula((location(l1)&location(l2))):
     n-or{pathoflength(l1,l2,u)|
       u: last(u)},
       0
   );
pathoflength(l1,l2,v)=
(sformula((location(l1)&location(l2))):
  (sformula(zero(v)):
    sformula(l1=l2),
     (n-or{pathoflength(l1,l2,u)
           |u:pred(u,v)
         }:
1,
         n-or{
         @connectoneighbor(l1,l2,w,u)
 |u,w: (pred(u,v)&((leftof(w,l2)
 |leftof(l2,w))
 |(belowof(w,l2)|belowof(l2,w))))
 }
        )
   ),
   0
);

@connectoneighbor(l1,l2,ln,u)=
pathoflength(l1,ln,u):
  (connectionblocked(ln,l2):0,1),0);

% Two locations are not  connected
% if either contains a block

connectionblocked(l1,l2)=(
  sformula((
     (location(l1)&location(l2))
     &
     ((leftof(l1,l2)|leftof(l2,l1))
     |(belowof(l1,l2)|belowof(l2,l1))
     )
  )):
  n-or{blocks(u,l1),blocks(u,l2)
   |u:block(u)
  },
0
);
```

Acknowledgments

The authors would like to thank Nasa (contract NASA-NNG05GP48G) and Arda for supporting this research.

References

Balduccini, M.; and Gelfond, M. 2003. Logic programs with consistency-restoring rules. In AAAI Spring 2003 Symposium, 9-18.

Baral, C. 2003. Knowledge representation, reasoning and declarative problem solving. Cambridge University Press.

Baral, C.; Gelfond, M.; and Rushton, N. 2004. Probabilistic reasoning with answer sets. In Proc. LRNMR-07.

Chavira, M.; Darwiche, A.; and Jaeger, M. 2004. Compiling relational Bayesian networks for exact inference. Proceedings of the Second European Workshop on Probabilistic Graphical Models(PGM-2004), The Netherlands, 49-56.

Gelfond, M.; and Lifschitz, V. 1988. The stable model semantics for logic programming. R.Kowalski, K. Bowen(Eds.), Logic Programming: Proc. Fifth Internat. Conference and Symposium, 1070-1080.

Iwan, G.; and Lakemeyer, G. 2002. What observations really tell us. In CogRob02.

Jaeger, M. 1997. Relational Bayesian networks. UAI97.

Niemelä, I.; and Simons, P. 1997. Smodels – an implementation of the stable model and well-founded semantics for normal logic programs. In Dix, J, Furbach, U, and Nerode, A, editors, Proc. 4th international conference on Logic programming and non-monotonic reasoning, 420–429. Springer.

Pearl, J. 2000. Causality: models, reasoning and inference. Cambridge University Press.

Simons, P. 1999. Extending the stable models semantics with more expressive rules. In Proc. LPNMR-99, Lecture Notes in Artificial Intelligence(LNCS). Springer Verlag, Berlin.