Modelling Probabilistic Causation in Decision Making

Luís Moniz Pereira, Carroline Kencana Ramli

Abstract Humans know how to reason based on cause and effect, but cause and effect is not enough to draw conclusions due to the problem of imperfect information and uncertainty. To resolve these problems, humans reason combining causal models with probabilistic information. The theory that attempts to model both causality and probability is called probabilistic causation, better known as Causal Bayes Nets.

In this work we henceforth adopt a logic programming framework and methodology to model our functional description of Causal Bayes Nets, building on its many strengths and advantages to derive a consistent definition of its semantics. ACORDA is a declarative prospective logic programming which simulates human reasoning in multiple steps into the future. ACORDA itself is not equipped to deal with probabilistic theory. On the other hand, P-log is a declarative logic programming language that can be used to reason with probabilistic models. Integrated with P-log, ACORDA becomes ready to deal with uncertain problems that we face on a daily basis. We show how the integration between ACORDA and P-log has been accomplished, and we present cases of daily life examples that ACORDA can help people to reason about.

1 Introduction

Humans reason basically based on cause and effect. David Hume described causes as objects regularly followed by their effects [1]. Hume attempted to analyse causation in terms of invariable patterns of succession that are referred to as "regularity

Luís Moniz Pereira
CENTRIA, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal. e-mail: lmp@di.fct.unl.pt

Carroline Kencana Ramli
CENTRIA, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal. e-mail: carroline.kencana@googlemail.com

theories" of causation. The difficulty with regularity theories is that most causes are not invariably followed by their effects. For example, it is widely acceptable that smoking is a cause of lung cancer, but not all smokers have lung cancer. By contrast, the central idea behind probabilistic theories of causation is that causes raise the probability of their effects; an effect may still occur in the absence of a cause or fail to occur in its presence. The probabilistic theorem of causation helps in defining a pattern in problems with imperfect regularities [2]. This approach is called Causal Bayes Nets.

Translation of human reasoning using causal models and Bayes Nets into a computational framework is possible using logic programming. The main argument is that humans reason using logic. There is an obvious human capacity for understanding logic reasoning, one that might even be said to have developed throughout our evolution. Logic itself can be implemented on top of a symbol processing system like a computer. Thus, in this work we henceforth adopt a declarative logic programming framework and methodology to model our functional description of causal models and Bayes Nets, building on its many strengths and advantages to derive both a consistent definition of its semantics and a working implementation with which to conduct relevant experiments.

The paper is organized as follows: the next Section provides the background of prospective logic programming and probabilistic logic programming. Section 3 provides the explanation of our implementation and the paper continues with an example of how our implementation can be used in Sect. 4. The paper finishes with conclusions and directions for future work in Sect. 5.

2 Prospective and Probabilistic Logic Programming

2.1 Prospective Logic Programming

Prospective logic programming is an instance of an architecture for causal models, which implies a notion of simulation of causes and effects in order to solve the choice problem for alternative futures. This entails that the program is capable of conjuring up hypothetical *what-if* scenaria and formulating abductive explanations for both external and internal observations. Since we have multiple possible scenaria to choose from, we need some form of preference specification, which can be either a priori or a posteriori. A priori preferences are embedded in the program's own knowledge representation theory and can be used to produce the most relevant hypothetical abductions for a given state and observations, in order to conjecture possible future states. A posteriori preferences represent choice mechanisms, which enable the program to commit to one of the hypothetical scenaria engendered by the relevant abductive theories. These mechanisms may trigger additional simulations, by means of the functional connectivity, in order to posit which new information to

acquire, so more informed choices can be enacted, in particular by restricting and committing to some of the abductive explanations along the way.

Definition 1 (Language). Let $\mathcal{L}$ be a *first order language*. A domain literal in $\mathcal{L}$ is a domain atom *A* or its default negation *not A*, the latter expressing that the atom is false by default. A domain rule in $\mathcal{L}$ is a rule of the form:

$$A \leftarrow L_1, \ldots, L_t \ \ (t \geq 0)$$

where *A* is a domain atom and $L_1, \ldots, L_t$ are domain literals.

Definition 2 (Integrity Constraint). An *integrity constraint* in $\mathcal{L}$ has a form:

$$\bot \leftarrow L_1, \ldots, L_t \ \ (t > 0)$$

where $\bot$ is a domain atom denoting falsity, and $L_1, \ldots, L_t$ are domain literals.

A *(logic) program* P over $\mathcal{L}$ is a set of domain rules and integrity constraints, standing for all their ground instances.

Each program P is associated with a *set of abducibles* $\mathcal{A}_P \subseteq \mathcal{L}$. Abducibles can be seen as hypotheses that provide hypothetical solutions or possible explanations of given queries. An abducible a can be assumed in the program only if it is a considered one, i.e. if it is expected in the given situation, and moreover there is no expectation to the contrary [3]. The atom $consider(a)$ will be true if and only if the abducible a is considered. We can define it by the logic programming rule:

$$consider(A) \leftarrow expect(A), not\ expect\_not(A)$$

where A stands for a logic programming variable. The rules about expectations are domain-specific knowledge contained in the theory of the program, and effectively constrain available the hypotheses .

To express preference criteria amongst abducibles, we introduce the language $\mathcal{L}^*$. Let $\mathcal{L}^*$ be a language consisting of logic program and relevance rules.

Definition 3 (Relevance Rule). Let a and b are abducibles. A *relevance atom* $a \lhd b$ means abducible a is more relevant or preferred than abducible b, i.e. one cannot have b without also having a. A *relevance rule* is one of the form:

$$a \lhd b \leftarrow L_1, \ldots, L_t \ \ (t \geq 0)$$

where $a \lhd b$ is a relevance atom and every $L_i (1 \leq i \leq t)$ is a domain literal or a relevance literal.

Example 1 (Tea 1). Consider a situation where an agent Claire drinks either tea or coffee (but not both). She does not expect to have coffee if she has high blood pressure. Also suppose that agent Claire prefers coffee over tea when she is sleepy. This situation can be represented by a program Q with the set of abducibles $\mathcal{A}_Q = \{tea,\ coffee\}$:

```
1 falsum ← not drink.
2 drink ← consider(tea).
3 drink ← consider(coffee).
4 constrain(1,[tea,coffee],1).
5 expect(tea). expect(coffee).
6 expect_not(coffee) ← blood_pressure_high.
7 coffee ◁ tea ← sleepy.
```

The query is triggered by an integrity constraint coded with `falsum/0`. ACORDA has to fulfill the integrity constraint by trying to find all the explanations to the atom $drink$. We codified the preference using the `</2` built–in ACORDA predicate. The exclusivity of the two abducibles is coded with `constrain/3` on line 4. So there are two abductive solusions: $A_1 = \{coffee\}$, $A_2 = \{tea\}$. We should choose our solution from those two explanations based on our preferences. If we assert $sleepy$ is true, the only explanation left is A_1 because of the preference rule – in order to have the abducible tea we should have the abducible $coffee$. On the other hand, if we assert $blood\_pressure\_high$ is true, the only remaining solution is A_2 because the abducible $coffee$ becomes not expected.

Having the notion of expectation allows one to express the preconditions for an expectation or otherwise about an abducible a, and expresses which possible expectations are confirmed (or assumed) in a given situation. If the preconditions do not hold, then abducible a cannot be considered, and therefore a will never be assumed. By means of `expect_not/1` one can express situations where one does not expect something. In this case, when blood pressure is high, coffee will not be considered or assumed because of the contrary expectation arising as well (and therefore tea will be assumed).

2.2 Probabilistic Logic Programming

Probabilistic logic programming (P-log) was introduced for the first time by Chitta Baral et.al [4]. P-log is a declarative language that combines logical and probabilistic reasoning, and uses Answer Set Programming (ASP) as its logical foundation and Causal Bayes Nets [5] as its probabilistic foundation. P-log can also represent a mechanism for updating or reassessing probability [6].

The declaration part of a P-log program Π contains sorts and attributes. A sort c is a set of terms. It can be defined by listing all its elements: $c = \{x_1, x_2, \ldots, x_n\}$. Given 2 integers $L \leq U$ we can also use 2 shortcut notations: $c = \{L..U\}$ for the sort $c = \{L, L+1, \ldots, U\}$ and $c = \{h(L..U)\}$ for the sort $c = \{h(L), h(L+1), \ldots, h(U)\}$. We are also able to define a sort by arbitrarily mixing the previous constructions, e.g. $c = \{x_1, .., x_n, L..U, h(M..N)\}$. In addition, it is allowed to declare union as well as intersection of sorts. A union sort is represented by $c = union(c_1, ..., c_n)$ while an intersection sort by $c = intersection(c_1, ..., c_n)$, where $c_i, 1 \leq i \leq n$ are declared sorts.

Definition 4 (Sorted Signature). The *sorted signature* Σ of a program Π 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.

Definition 5 (Attribute). Let $c_0, c_1, \ldots, c_n$ be sorts. An *attribute* a with the domain $c_1 \times ... \times c_n$ and the range c_0 is represented as follows:

$$a : c_1 \times ... \times c_n \ \rightarrow \ c_0$$

If attribute a has no domain parameter, we simply write $a : c_0$. The range of attribute a is denoted by $range(a)$. Attribute terms are expressions of the form $a(\bar{t})$, where a is an attribute and $\bar{t}$ is a vector of terms of the sorts required by a.

The *regular part* of a P-log program Π consists of a collection of rules, facts and integrity constraints formed using literals of Σ.

Definition 6 (Random Selection Rule). A *random selection rule* has a form:

$$random(RandomName, a(\bar{t}),\ DynamicRange) \ \mathbf{:-}\ Condition$$

This means that the attribute instance $a(\bar{t})$ is random if the conditions in $Condition$ are satisfied. The $DynamicRange$ allows to restrict the default range for random attributes. The $RandomName$ is a syntactic mechanism used to link random attributes to the corresponding probabilities. The constant $full$ is used in $DynamicRange$ to signal that the dynamic domain is equal to $range(a)$.

Definition 7 (Probabilistic Information). Information about probabilities of random attribute instances $a(\bar{t})$ taking particular value y is given by probability atoms (or simply pa-atoms) which have the following form:

$$pa(RandomName, a(\bar{t}, y), d\_(A, B)) \ \mathbf{:-}\ Condition$$

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

Definition 8 (Observations and Actions). *Observations* and *actions* are, respectively, statements of the forms $obs(l)$ and $do(l)$, where l is a literal. Observations are used to record the outcomes of random events, i.e. random attributes and attributes dependent on them.

Example 2 (Tea 2). (Continuation of Example 1) Suppose if we know that the availability of tea for agent Claire is around 60%.

```
8  beginPr.
9      beverage = {tea, coffee}.
10     available : beverage.
11     random(rd, available, full).
12     pa(rd, available(tea), d_(60, 100)).
13 endPr.
```

The probabilistic part is coded in between `beginPr/0` and `endPr/0`. There are two kinds of beverage: tea and coffee and both are randomly available. We get the information the availability of tea is 60%, coded by $pa(rd, available(tea), d\_(60, 100))$.

3 Implementation

ACORDA [7, 8][1] is a system that implements prospective logic programming. ACORDA is the main component of our system with P-log used as probabilistic support in the background. Computation in each component is done independently but they can cooperate in providing the information needed. Both ACORDA and P-log rely on the well-known Stable Model semantics to provide meaning to programs (see their references for details).

The system architecture follows this separation in a very explicit way. There is indeed a necessary separation between producer and consumer of information. ACORDA and P-log can both act as a producer or consumer of information. The interfaces between the various components of the integration of ACORDA with P-log are made explicit in Fig. 1[2]. Each agent is equipped with a Knowledge Base and a Bayes Net as its initial theory. The first time around, ACORDA sends Bayes Net information to P-log and later P-log translates all the information sent by ACORDA and keeps it for future computation. The problem of prospection is then that of finding abductive extensions to this initial theory which are both relevant (under the agent's current goals) and preferred (w.r.t. the preference rules in its initial theory). The first step is to select the goals that the agent will possibly attend to during the prospective cycle. Integrity constraints are also considered to ensure the agent always performs transitions into valid evolution states.

Once the set of active goals for the current state is known, the next step is to find out which are the relevant abductive hypotheses. At this step, P-log uses the abductive hypothesis for computing probabilistic information to help ACORDA do a priori preferences. Forward reasoning can then be applied to abducibles in those scenaria to obtain relevant side-effect consequences, which can then be used to enact a posteriori preferences. These preferences can be enforced by employing utility theory that can be combined with probabilistic theory in P-log. In case additional information is needed to enact preferences, the agent may consult external oracles. This greatly benefits agents in giving them the ability to probe the outside environment, thus providing better informed choices, including the making of experiments. Each oracle mechanism may have certain conditions specifying whether it is available for questioning. Whenever the agent acquires additional information, it is possible that ensuing side-effects affect its original search, e.g. some already consid-

[1] The integration of ACORDA with P-log can be accessed at http://sites.google.com/site/acordaplog/Home

[2] Please note the dashed lines represent communication between ACORDA and P-log.

ered abducibles may now be disconfirmed and some new abducibles are triggered. To account for all possible side-effects, a second round of prospection takes place.

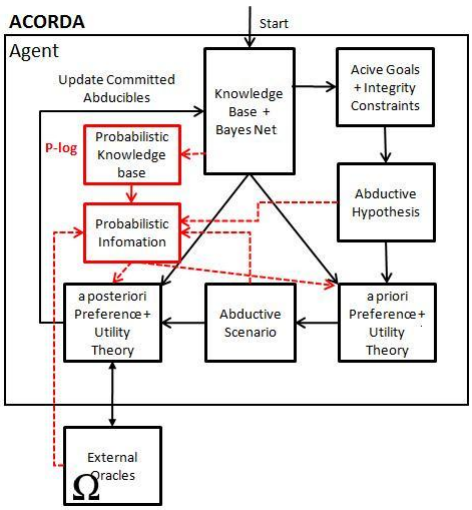

Fig. 1: Integration Architecture

If everything goes well and only a single model emerges from computation of the abductive stable models, the ACORDA cycle terminates and the resulting abducibles are added to the next state of the knowledge base. In most cases, however, we cannot guarantee the emergence of a single model, since the active preferences may not be sufficient to defeat enough abducibles. In these situations, the ACORDA system has to resort on additional information for making further choices. If no model emerges, relaxation of constraints and preferences may be in order.

Indeed, a given abducible can be defeated in any one of two cases: either by satisfaction of an `expect_not/1` rule for that abducible, or by satisfaction of a preference rule that prefers another abducible instead. However, the current knowledge state may be insufficient to satisfy any of these cases for all abducibles except one, or else a single model would have already been abduced. It is then necessary that the system obtains the answers it needs from somewhere else, namely from making experiments on the environment or from querying an outside entity.

ACORDA consequently activates its a posteriori choice mechanisms by attempting to satisfy additional selecting preferences. There are two steps: first ACORDA computes the utility value for each abducible. ACORDA selects amongst them based on the preference function defined. This step is coded below in the meta predicate `select/2`.

```
1 select(M, NewM) :- select1(M, M1), select2(M1, NewM).
2 select1(M, M1)  :- addUtilityValue(M, M1).
3 select2(M1, NewM) :-%use preference function to select the model.
```

Example 3 (Tea 3). (Continuation of Example 2) Consider now we introduce the utility rate for coffee and tea for agent Claire based on her preference. What do we suggest for agent Claire's beverage when the utility probability value is taken into account?

```
14 beginProlog.
15    select(M, Mnew) :-
16       utilityRate(tea, 0.8), utilityRate(coffee, 0.7),
17       select1(M, M2),
18       select2(M2, 0, [], Mnew).
19 % Add utility value to each model
20    select1([], []).
21    select1([X|Xs], [Y|Ys]) :-
22       addUtilityValue(X, Y), select1(Xs, Ys).
23 % A posteriori preference models
24    select2([], _, M, M).
25    select2([M|Ms], Acc, OldM, NewM) :-
26       member(utilityModel(U), M),
27       U > Acc → select2(Ms,U,M,NewM); select2(Ms,Acc,OldM,NewM).
28    addUtilityValue([X], [utilityModel(UModel)|[X]]) :-
29       holds utilityRate(X, R),
30       pr(available(X), P), UModel is R * P.
31    addUtilityValue(_, []).
32 endProlog.
```

First, ACORDA launches oracles to acquire information about Claire's condition – whether she is sleepy and whether she has high blood pressure. If there is no contrary expectation for any of the beverages, i.e. agent Claire is not sleepy and also does not have high blood pressure, we will have two different abductive solutions: $M_1 = \{coffee\}$, $M_2 = \{tea\}$. Next, ACORDA performs a posteriori selection. Our selecting preference amongst abducibles is codified on lines 14-32. First we initialize the utility rate for both beverages. In the `addUtilityValue/2` predicate, we define our utility function. In this example, we define the utility value for each beverage as its probability of availability times its utility rate. After the computation we get the result: $M_1 = \{utilityModel(0.2800),\ coffee\}$, $M_2 = \{utilityModel(0.4800),\ tea\}$. Predicate `select2/2` will select the highest utility model and the final result is $M_2 = \{utilityModel(0.4800),\ tea\}$ which means that based on the a posteriori preference using our utility function, agent Claire is encouraged to have tea.

4 Example: Risk Analysis

The economics of risk[9] has been a fascinating area of inquiry for at least two reasons. First, there is hardly any situation where economic decisions are made with perfect certainty. The sources of uncertainty are multiple and pervasive. They include price risk, income risk, weather risk, health risk, etc. As a result, both private and public decisions under risk are of considerable interest. This is true in positive

analysis (where we want to understand human behaviour), as well as in normative analysis (where we want to make recommendations about particular management or policy decisions). Second, over the last few decades, significant progress has been made in understanding human behaviour under uncertainty. As a result, we have now a somewhat refined framework to analyse decision-making under risk.

In a sense, the economics of risk is a difficult subject; it involves understanding human decisions in the absence of perfect information. In addition, we do not understand well how the human brain processes information. As a result, proposing an analytical framework to represent what we do not know seems to be an impossible task. In spite of these difficulties, much progress has been made. First, probability theory is the cornerstone of risk assessment. This allows us to measure risk in a fashion that can be communicated amongst decision makers or researchers. Second, risk preferences are better understood. This provides useful insights into the economic rationality of decision-making under uncertainty. Third, over the last decades, good insights have been developed about the value of information. This helps us to better understand the role of information and risk in private as well as public decision-making.

We define risk as representing any situation where some events are not known with certainty. This means that one cannot influence the prospects for the risk. It can also relate to events that are relatively rare. The list of risky events is thus extremely long. First, this creates a significant challenge to measure risky events. Indeed, how can we measure what we do not know for sure? Second, given that the number of risky events is very large, is it realistic to think that risk can be measured? We will present a simple example about decision–making in a restaurant where risk is taken into account.

Example 4 (Mamamia). The owner of the Restaurant "Mamamia" wants to offer a new menu. Before the launch of the new menu, he performed some research. Based on it, 75% of teenagers prefer the new menu and 80% of adults prefer the original menu. Around 40% of his costumers are teenagers. If he offers the new menu, each new menu will return 5 Euros. The basic cost that he should spend for 100 new menus is 200 Euros. What is your suggestion for the owner of the Restaurant "Mamamia"? The owner's utility function is approximately represented by $U(X) = 2 * X - 0.01X^2$, $(X \leq 100)$, X being his income.

```
1  expect(decide(launch_new_menu)). expect(decide(not_launch)).
2  constrain(1,[decide(launch_new_menu), decide(not_launch)], 1).
3  decision ← consider(decide(X)).
4  decide(launch_new_menu) ◁ decide(not_launch) ← not_take_risk,
5       prolog(pr(menu(new), PN)), prolog(pr(menu(original), PO)),
6       prolog(PN > PO).
7  decide(not_launch) ◁ decide(launch_new_menu) ← not_take_risk,
8       prolog(pr(menu(original), PO)), prolog(pr(menu(new), PN)),
9       prolog(PO > PN).
10 not_take_risk ← not take_risk.
11 falsum ← not decision.
12 beginPr.
13    age = {teenager, adult}.    offer = {original, new}.
```

```
14    customer : age.
15    random(rc, customer, full).
16    pa(rc, customer(teenager), d_(60,100)).
17    menu : offer.
18    random(ro, menu, full).
19    pa(ro, menu(new), d_(75,100)) :- customer(teenager).
20    pa(ro, menu(original), d_(80,100)) :- customer(adult).
21 endPr.
22 beginProlog.
23    :- import member/2, length/2 from basics.
24    select(M, Mnew) :- select1(M, Mnew), select2(_,0,[],_).
25    select1([],[]).
26    select1([X|Xs], [Y|Ys]) :-
27       addUtilityValue(X,Y), select1(Xs,Ys).
28    select2([],_,M,M).
29    select2([M|Ms], Acc,OldM,NewM):-
30       member(utilityModel(U), M),
31       U > Acc → select2(Ms,U,M,NewM); select2(Ms,Acc,OldM,NewM).
32    addUtilityValue([decide(X)],
33       [utilityModel(EU), expectedProfit(Profit),decide(X)]) :-
34          expectedProfit(X, Profit), expectedUtility(X, EU).
35    expectedProfit(Action, P) :-
36       return(Action, R), cost(Action, C), P is R − C.
37    expectedUtility(Action, EU) :-
38       pr(menu(new), PrN), expectedProfit(Action, Pf),
39       EU is (2∗Pf∗PrN − 0.01∗Pf∗Pf∗PrN).
40    return(launch_new_menu, 5). return(not_launch, 0).
41    cost(launch_new_menu, 2).   cost(not_launch, 0).
42 endProlog.
```

In Example 4, the expected return value depends on the probability of the number of new menu offers. Given probability information about the age of customers and their behaviour in choosing a menu offer, we compute the expected probability of new menu choices. In this case, the probabilty of new menu choices is 0.42 an the probability of original menu choices is 0.58. The probability of original menu choices is higher than the probability of new menu choices. Furthermore, if we compute the expected return more comprehensively, we will get the result:
$M_1 = \{utilityModel(3.1323),\ expectedProfit(3),\ decide(launch\_new\_menu)\}$,
$M_2 = \{utilityModel(0.0000),\ expectedProfit(0),\ decide(not\_launch)\}$
Based on the computation, it is better if he launches a new menu even though it is risky.

5 Conclusion and Future Work

Humans reason using cause and effect models with the combination of probabilistic models such as Bayes Nets. Unfortunately, the theory of Bayes Nets does not provide tools for generating a scenario that is needed for generating several possible worlds. Those are important in order to simulate human reasoning not only about the

present but also about the future. On the other hand, ACORDA is a prospective logic programming language that is able to prospect possible future worlds based on abduction, preference and expectation specifications. But ACORDA itself cannot handle probabilistic information. To deal with this problem, we integrated ACORDA with P-log, a declarative logic programming language based on probability theory and Causal Bayes Nets. Now, using our new system, it is easy to create models using both causal models and Bayes Nets. The resulting declarative system can benefit from the capabilities of the original ACORDA for generating scenaria, and is equipped with probabilistic theory as a medium to handle uncertain events. Using probability theory and utility functions, the new ACORDA is now more powerful in managing quantitative as well as qualitative a priori and a posteriori preferences.

Often we face situations when new information is available that can be added to our model in order to perform our reasoning better. Our new ACORDA also makes it possible to perform a simulation and afterwards add more information directly to the agent. It can also perform several steps of prospection in order to predict the future better. For illustrative understanding, we presented taking a risk example in which the system can help people to reason and rationally decide.

Applying utility functions to decision-making under uncertainty requires having good information about the measurement of the probability disribution and the risk preferences of the decision-maker. It is possible to conduct risk analysis without precise information about risk preferences using stochastic dominance. The latter sees the elimination of "inferior choices" without strong a priori information about risk preferences. For future work, we can extend P-log to handle stochastic processes.

References

1. D. Hume. *An Enquiry Concerning Human Understanding: A Critical Edition*. Oxford Philosophical Texts, 1748.
2. J. Mackie. *The Cement of the Universe*. Oxford: Clarendon Press, 1974.
3. P. Dell'Acqua and L. M. Pereira. Preferential theory revision (extended version). *J. Applied Logic*, 2007.
4. C. Baral, M. Gelfond, and N. Rushton. Probabilistic reasoning with answer sets. In *LPNMR7*, pages 21–33, 2004.
5. J. Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2000.
6. M. Gelfond, N. Rushton, and W. Zhu. Combining logical and probabilistic reasoning. In *AAAI Spring Symposium*, 2006.
7. L. M. Pereira and G. Lopes. Prospective logic agents. In *AI Procs. 13th Portuguese Intl.Conf. on Artificial Intelligence (EPIA'07)*, pages 73–86, 2007.
8. L. M. Pereira and G. Lopes. Prospective logic agents. In *International Journal of Reasoning-based Intelligent Systems (IJRIS)*, to appear in 2009.
9. Jean-Paul Chavas. *Risk Analysis in Theory and Practice*. Academic Press, 2004.