

Modelling Decision Making with Probabilistic Causation

Luís Moniz Pereira and Carroline Kencana Ramli *

June 11, 2009

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 system 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.

Keywords: P-log, ACORDA, Prospective Logic Programming, Human Reasoning, Causal Models, Bayes Nets.

*Centro de Inteligência Artificial (CENTRIA), Universidade Nova de Lisboa, 2829-516 Caparica, Portugal, Email: imp@di.fct.unl.pt, carroline.kencana@gmail.com

1 Introduction

Studying how human brain works [6] is one of the most astonishing areas of research. It draws from studies of intelligence by psychologists, but even more from ethology, evolutionary biology, linguistics, and the neurosciences [29, 9, 12, 11, 18]. The computational aspects of human brain are one of the attractive research areas of computer science. Nowadays, a lot of research in the Artificial Intelligence (AI) community tries to mimic how humans reason.

Basically humans reason based on cause and effect. David Hume described causes as objects regularly followed by their effects [15]

We may define a cause to be an object, followed by another, and where all the objects similar to the first, are followed by objects similar to the second.

Hume attempted to analyse causation in terms of invariable patterns of succession that are referred to as “regularity theories” of causation. There are a number of well-known difficulties with regularity theories, and these may be used to motivate probabilistic approaches to causation.

The difficult part in regularity theories is that most causes are not invariably followed by their effects. For example, it’s 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 [17].

If A causes B , then, typically, B will not also cause A . Smoking causes lung cancer, but lung cancer does not cause someone to smoke. In other words, causation is usually asymmetric, cause and effect can not be commuted. In addition, the asymmetry of causal relation is unrelated with the asymmetry of causal implication and its contraposition [27]. For example, if A stands for the statement “Peter smokes” and B stands for the statement “Peter has lung cancer”, then A implies B (smoking causes lung cancer) but $\neg B$ doesn’t imply $\neg A$ (the absence of lung cancer doesn’t cause Peter not to smoke). This may pose a problem for regularity theories. It would be nice if a theory of causation could provide some explanation of the directionality of causation, rather than merely stipulate it [16].

For a few decades, statisticians, computer scientists and philosophers have worked on developing a theory about how to represent causal relations and how causal claims connect with probabilities. Those representations show how information about some features of the world may be used to compute probabilities for other features, how partial causal knowledge may be used to compute the effects of actions and how causal relations can be reliably learned, at least by computers. The resulting theory is called Causal Bayes Nets.

Translation of human reasoning using causal models and Bayes Nets described previously into a computational framework would be possible using logic programming. The main argument is that humans reason using logic. Logic itself can be implemented on top of a symbol processing system like a computer. On the other hand, there is an obvious human capacity for understanding logic reasoning, one that might even be said to have developed throughout our evolution. Its most powerful expression today is science itself, and the knowledge amassed from numerous disciplines, each with its own logic. From state laws to quantum physics, logic has become the foundation on which human knowledge is built and improved.

A part of the AI community has struggled, for some time now, to turn logic into an effective programming language, allowing it to be used as a system and application specification language which is not only executable, but on top of which one can demon-

strate properties and proofs of correctness that validate the very self-describing programs which are produced. At the same time, AI has developed logic beyond the confines of monotonic cumulativity and into the non-monotonic realms that are typical of the real world of incomplete, contradictory, arguable, revised, distributed and evolving knowledge. Over the years, enormous amount of work and results have been achieved on separate topics in logic programming, language semantics, revision, preferences, and evolving programs with updates [10, 25, 1]. Computational logic has shown itself capable to evolve and meet the demands of the difficult descriptions it is trying to address.

Thus, in this work we henceforth adopt a 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 use of the logic paradigm also allows us to present the discussion of our system at a sufficiently high level of abstraction and generality to allow for productive interdisciplinary discussions both about its specification and the derived properties. The language of logic is universally used by both natural sciences and humanities, and more generally at the core of any source of human derived knowledge, so it provides us with a common ground on which to reason about our theory. Since the field of cognitive science is essentially a joint effort on the part of many different kinds of knowledge fields, we believe such language and vocabulary unification efforts are not only useful but mandatory.

The paper is organized as follows: the next two Sections provide the background of prospective logic programming and probabilistic logic programming. Section 4 provides the explanation of our implementation and the paper continues with an example of how our implementation can be used in Section 5. The paper finishes with conclusions and directions for future work in Section 6.

2 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.

2.1 Language

Definition 2.1 (Language) *Let \mathcal{L} be a first order language. A domain literal in \mathcal{L} is a domain atom A or its default negation $\text{not } A$, the latter expressing that the atom is false by default. A domain rule in \mathcal{L} is a rule of the form:*

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

where A is a domain atom and L_1, \dots, L_t are domain literals.

Definition 2.2 (Integrity Constraint) *An integrity constraint in \mathcal{L} has a form:*

$$\perp \leftarrow L_1, \dots, L_t \quad (t > 0)$$

where \perp is a domain atom denoting falsity, and L_1, \dots, 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.

2.2 Abducibles

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 [10]. The atom $\text{consider}(a)$ will be true if and only if the abducible a is considered. We can define it by the logic programming rule:

$$\text{consider}(A) \leftarrow \text{expect}(A), \text{not } \text{expect_not}(A)$$

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.

2.3 Preferring Abducibles

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 2.3 (Relevance Rule) *Let a and b be abducibles. A relevance atom $a \triangleleft b$ means abducible a is more relevant or preferred than abducible b , i.e. one can not abduce b without also abducting a . A relevance rule is one of the form:*

$$a \triangleleft b \leftarrow L_1, \dots, L_t \quad (t \geq 0)$$

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

Example 2.4 (Tea 1, taken from [21])

Consider a situation where Claire drinks either tea or coffee (but not both). Suppose that Claire prefers coffee over tea when she wants to keep herself awake, and doesn't drink coffee when she has high blood pressure. Moreover, if she wants to

socialize, then she prefers drinking tea over coffee; and she prefers socializing over staying wide awake if she is in the mood for it. If Claire is sleepy, then she expects to wake up, and she expects to socialize if she is in the mood for it. This situation is described by program P over \mathcal{L} with abducibles $A_P = \{tea, coffee, awake, socialize\}$:

1. $drink \leftarrow tea.$
 $drink \leftarrow coffee.$
2. $expect(tea).$
 $expect(coffee).$
 $expect(socialize) \leftarrow in_the_mood.$
 $expect(awake) \leftarrow sleepy.$
3. $expect_not(coffee) \leftarrow$
 $blood_pressure_high.$
4. $constrain(0, [tea, coffee], 1).$
5. $coffee \triangleleft tea \leftarrow awake.$
6. $tea \triangleleft coffee \leftarrow socialize.$
7. $socialize \triangleleft awake \leftarrow in_the_mood.$

In the abductive stable model semantics, this program has two models, one with *tea* the other with *coffee*. Adding literal *sleepy*, enforced abduction comes into play, enabling the preference of *coffee* over *tea* given *awake*, thus defeating the abductive stable model where only *tea* is present (due to the impossibility of simultaneously abducting *coffee*). If later we add *blood_pressure_high*, *coffee* is no longer expected, and the transformed preference rule no longer defeats the abduction of *tea* which then becomes the single abductive stable model, despite the presence of *sleepy*.

Moreover, if instead of adding *blood_pressure_high* we added *in_the_mood*, the preference of *socialize* over *awake* would be triggered. In this scenario, now the preference of *tea* over *coffee* is enforced and the preference of *coffee* over *tea* is disabled. The preference of *socialize* over *awake*, enabled by the addition of *in_the_mood*, acts as a meta-preference conditioning which preference rule between *coffee* and *tea* would be enacted — even though *sleepy* is also present.

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 can not 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).

2.4 Abducibles Sets

In many situations it is desirable not only to include rules about the expectations for single abducibles, but also to express contextual information constraining the powerset of abducibles. For instance, in the previous example we expressed that abducting tea or coffee was mutually exclusive (i.e. only one of them could be abduced), but it is easy to imagine similar choice situations where it would be possible, indeed even desirable, to abduce both, or neither. The behaviour of abducibles over different sets is highly context-dependent, and as such, should also be embedded over rules in the theory.

Overall, the problem is analogous to the ones addressed by cardinality and weight constraint rules for the Stable Model semantics, and below we present how one can nicely import these results to work with abduction of sets, and also hierarchies of sets.

Example 2.5 Consider a situation where Claire is deciding what to have for a meal from a limited buffet. The menu has appetizers (which Claire doesn't mind skipping, unless she's very hungry), three main dishes, from which one can select a maximum of two, and drinks, from which she will have a single one. The situation, with all possible choices, can be modelled by the following program P over \mathcal{L} with the set of abducibles

$$A_P = \{bread, salad, cheese, fish, meat, veggie, wine, juice, water, appetizers, drinks, main_dishes\}$$

1. $constrain(0, [bread, salad, cheese], 3) \leftarrow$
 $appetizers.$
2. $constrain(1, [fish, meat, veggie], 2) \leftarrow$
 $main_dishes.$
3. $constrain(1, [wine, juice, water], 1) \leftarrow$
 $drinks.$

4. `constrain(2, [asppetizers, main_dishes, drinks], 3).`
5. `main_dishes < appetizers.`
6. `drinks < appetizers.`
7. `appetizers ← very_hungry.`

In this situation we model appetizers as being the least preferred set from those available for the meal. This shows how we can condition sets of abducibles based on the generation of literals from other cardinality constraints along with preferences amongst such literals.

2.5 A Posteriori Preference

Abduction can be seen as a mechanism to enable the generation of the possible futures of an agent, with each abductive stable model representing a possibly reachable scenario of interest. Preferring over abducibles enacts preferences over the imagined future of the agent. In this context, it is unavoidable to deal with uncertainty, a problem decision theory is ready to address using probabilities coupled with utility functions.

Example 2.6 *Suppose Claire is spending a day at the beach and she is deciding what means of transportation to adopt. She knows it is usually faster and more comfortable to go by car, but she also knows, because it is hot, there is possibility of a traffic jam. It is also possible to use public transportation (by train), but it will take longer, though it meets her wishes of being more environment friendly. The situation can be modeled by the abductive logic program:*

1. `hot.`
2. `go_to(beach) ← car.`
`go_to(beach) ← train.`
3. `expect(car).`
`expect(train).`
4. `constraint(1, [car, train], 1).`
5. `probability(traffic_jam, 0.7) ← hot.`
`probability(not traffic_jam, 0.3) ← hot.`
6. `utility(stuck_in_traffic, -8).`
`utility(wasting_time, -4).`
`utility(comfort, 10).`
`utility(environment_friendly, 3).`

By assuming each of the abductive hypotheses, the general utility of going to the beach can be computed for each particular scenario:

Assume car

Probability of being stuck in traffic = 0.7

Probability of a comfortable ride = 0.3

*Expected utility = $10 * 0.3 + 0.7 * -8 = -2.6$*

Assume train

Expected utility = $-4 + 3 = -1$

It should be clear that enacting preferential reasoning over the utilities computed for each model has to be performed after the scenarios are available, with an *a posteriori* meta-reasoning over the models and their respective utilities.

Once each possible scenario is actually obtained, there are a number of different strategies which can be used to choose which of the scenaria leads to more favourable consequences. A possible way to achieve this is using numeric functions to generate a quantitative measure of utility for each possible action. We allow for the application of this strategy, by making a priori assignments of probability values to uncertain literals and utilities to relevant consequences of abducibles. We can then obtain a posteriori the overall utility of a model by weighing the utility of its consequences by the probability of its uncertain literals. It is then possible to use this numerical assessment to establish a preorder amongst remaining models.

Both qualitative and quantitative evaluations of the scenaria can be greatly improved by merely acquiring additional information to make a final decision. We next consider the mechanism that our agents use to question external systems, be they other agents, actuators, sensors or other procedures. Each of these serves the purpose of an oracle, which the agent can probe through observations of its own.

Having computed possible scenaria, represented by abductive stable models, more favourable scenaria can be preferred amongst them a posteriori. Typically, a posteriori preferences are performed by evaluating consequences of abducibles in abductive stable models. The evaluation can be done quantitatively (for instance by utility functions) or qualitatively (for instance by enforcing some rules to hold). When

currently available knowledge is insufficient to prefer amongst abductive stable models, additional information can be gathered, e.g. by performing experiments or consulting an oracle. To realize a posteriori preferences, ACORDA provides predicate `select/2` that can be defined by users following some domain-specific mechanism for selecting favoured abductive stable models. The use of this predicate to perform a posteriori preferences will be discussed in a subsequent section.

3 Probabilistic Logic Programming

Probabilistic logic programming (P-log) was introduced for the first time by Chitta Baral et.al [3, 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 [19] as its probabilistic foundation.

The original P-log¹ [3, 4] uses Answer Set Programming (ASP) as a tool for computing all stable models of the logical part of P-log. Although ASP has been proved to be a useful paradigm for solving varieties of combinatorial problems, its non-relevance property [7] makes the P-log system sometimes computationally redundant. Newer developments of P-log² [2] use the XASP package of XSB Prolog for interfacing with an answer set solver.

The power of ASP allows the representation of both classical and default negation in P-log easily. Moreover, P-log(XSB) uses XSB as the underlying processing platform, allowing arbitrary Prolog code for recursive definitions. P-log can also represent a mechanism for updating or reassessing probability [13]. Later by using XASP, we can easily integrate with the prospective logic programming system ACORDA [24] that will be explained in Section 4.

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:

¹The original P-log can be accessed at <http://www.cs.ttu.edu/~wezhu/>

²The newer P-log(XSB) can be accessed at <http://sites.google.com/site/plogxsb/Home>

$c = \{x_1, x_2, \dots, x_n\}$. Given 2 integers $L \leq U$ we can also use 2 shortcut notations: $c = \{L..U\}$ for the sort $c = \{L, L+1, \dots, U\}$ and $c = \{h(L..U)\}$ for the sort $c = \{h(L), h(L+1), \dots, h(U)\}$. We are also able to define a sort by arbitrarily mixing the previous constructions, e.g. $c = \{x_1, \dots, x_n, L..U, h(M..N)\}$. In addition, it is allowed to declare union as well as intersection of sorts. A union sort is represented by $c = \text{union}(c_1, \dots, c_n)$ while an intersection sort by $c = \text{intersection}(c_1, \dots, c_n)$, where $c_i, 1 \leq i \leq n$ are declared sorts.

Definition 3.1 (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 3.2 (Attribute) *Let c_0, c_1, \dots, c_n be sorts. An attribute a with the domain $c_1 \times \dots \times c_n$ and the range c_0 is represented as follows:*

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

If attribute a has no domain parameter, we simply write $a : c_0$. The range of attribute a is denoted by $\text{range}(a)$. 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 3.3 (Random Selection Rule) *A random selection rule has a form:*

$\text{random}(\text{RuleName}, a(\bar{t}), \text{DynamicRange}) :- \text{Cond}$

This means that the attribute instance $a(\bar{t})$ is random if the conditions in *Cond* are satisfied. The *DynamicRange* allows to restrict the default range for random attributes. The *RuleName* 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 $\text{range}(a)$.

Definition 3.4 (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:

$$pa(\text{RuleName}, a(\bar{t}, y), d_-(A, B)) :- \text{Cond}$$

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

Example 3.5 (Tea 2) (*Continuation of Example 2.4*) 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))`.

Definition 3.6 (Observations and Actions)

Observations and actions are, respectively, statements of the forms $obs(l)$ and $do(l)$, where l is a literal. Observations are used to record the outcomes of random events, i.e. random attributes and attributes dependent on them. The statement $do(a(t, y))$ indicates that $a(t) = y$ is made true as the result of a deliberate (non-random) action.

Back to Example 3.5 as an illustration for Definition 3.6. The statement $obs(available(tea))$ indicates that we want to record the outcome of the availability of tea, on another hand the statement $do(available(tea))$ means *tea* was simply put on the table in the described action that tea is really available and we have tea already.

4 Implementation

ACORDA [22, 23]³ 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 [14] 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. 1⁴. 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

³The integration of ACORDA with P-log can be accessed at <http://sites.google.com/site/acordaplog/Home>

⁴Please note the dashed lines represent communication between ACORDA and P-log.

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 considered 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.

4.1 Expected Abducibles

Each abducible needs first to be expected (i.e. made available) by a model of the current knowledge state. This is achieved via the `expect/1` and `expect_not/1` clauses which indicate conditions under which an expectable abducible is indeed expected for an observation given the current knowledge state.

Sometimes we are not really sure about the relevant expected abducibles. What we do is to make an approximation of the expectation of some abducibles. We expect only those abducibles for which we reach a certain degree of belief. ACORDA handles this using probabilistic information as the pre-condition that must be satisfied before continuing the computation. This mechanism removes all unnecessary abducibles (i.e. all abducibles with the degree of belief below some particular value).

Example 4.1 (Cab 1) *There was an accident on the street that involved a cab. Two cab companies operate in the city: Blue and Green. Suppose the probability of a Green cab being involved is 70%. Now the police try to catch the driver. Who are the suspects?*

1. `falsum` \leftarrow `not catch_person`.
2. `catch_person` \leftarrow `consider(suspect(P))`.
3. `expect(suspect(P))` \leftarrow
`driver(P, C),`
`cab_company_involved(C, PR),`
`prolog(PR > 0.5)`.
4. `expect_not(suspect(P))` \leftarrow
`have_alibi(person(P))`.
5. `driver(antonio, blue)`.
`driver(berry, blue)`.

- `driver(charlie, blue)`.
6. `driver(peter, green)`.
`driver(robert, green)`.
`driver(john, green)`.
7. `have_alibi(person(P))` \leftarrow
`observe(prog, alibiOracle(P),`
`street_abc, true)`.
8. `observe(prog, alibiOracle(P), Q, R)` \leftarrow
`oracle,`
`prolog(oracleQuery(was_not_at(P, Q), R))`.
9. `cab_company_involved(C, PR)` \leftarrow
`cab(C), prolog(pr(cab(C), PR))`.
10. `cab(green)`. `cab(blue)`.
11. **beginPr**.
12. `color = {green, blue}`.
13. `cab : color`.
14. `random(rc, cab, full)`.
15. `pa(rc, cab(green), d_(70, 100))`.
16. **endPr**.

The abducible literals for this case are

$$Abs = \{suspect(antonio), suspect(berry), \\ suspect(charlie), suspect(john), \\ suspect(peter), suspect(robert)\}$$

There are six possible suspects and hence there are 63 (or $2^6 - 1$) non-empty subsets of the set of all possible suspects. On line 4 we state that someone is expected to be a suspect if he is one of the drivers of a cab company which is suspected to be involved in the accident by the degree of belief higher than 50%. At this stage, ACORDA calls P-log to find out the probability of each company being involved in the accident. Since the results are: `cab_company_involved(blue, 0.30)` and `cab_company_involved(green, 0.70)`, the only expected suspects are Peter, John and Robert, since they work for the Green company (line 6). Hence from 63 possibilities only 7 (or $2^3 - 1$) are remaining and it can make the computation much more efficient.

In the next step, the police can question the remaining possible suspects about their alibi, which is modelled as an external oracle. Based on the information provided, the set of expected suspects is finally identified.

4.2 Utility Function

Abduction can also be seen as a mechanism to enable the generation of the possible futures of an agent,

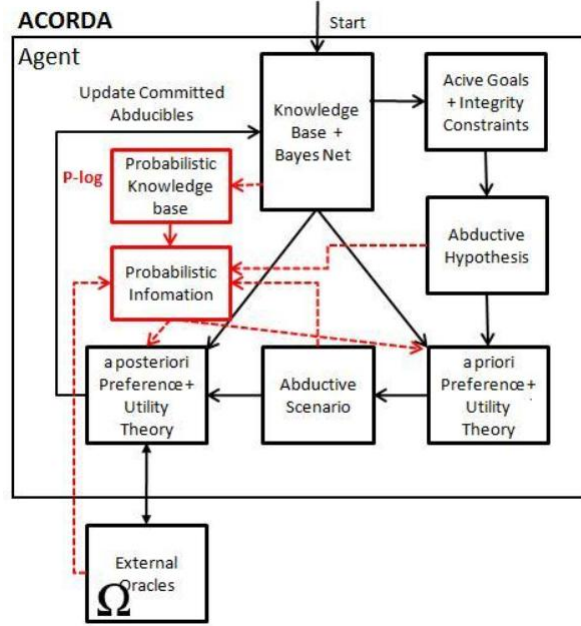


Figure 1: Integration Architecture

with each abductive stable model representing a possibly reachable scenario of interest. Preferring over abducibles in this case is enacting preferences over the imagined future of the agent. In this particular domain, it is unavoidable to deal with uncertainty, a problem that decision theory is ready to address using probability theory coupled with utility functions [5].

Example 4.2 (Cab 2) (Continuation of the Example 4.1) After finding the suspects and interrogating them, the police make a calculation and weight who has the highest chance of being a suspect. The fact that the accident happened in the night.

```

17. catch_person ← suspect(P,U).
18. suspect(P,U) ← consider(suspect(P)),
    prolog(utilityValue(P,U)),
    prolog(U < 0).
19. beginProlog.
20. utilityValue(P,U) :-
    (shift(P, night) →
     Rate is 1; Rate is 0),
    historyRecord(P,HR),
    alcoholRate(P, AR),

```

```

    yearsOfExperience(P,YE),
    U is HR - (Rate * AR * 1 / YE).
21. shift(antonio, day).
    shift(berry, night).
    shift(charlie, day).
    shift(peter, night).
    shift(robert, night).
    shift(john, day).
22. historyRecord(antonio, 0.6).
    historyRecord(berry, 0.65).
    historyRecord(charlie, 0.7).
    historyRecord(peter, 0.55).
    historyRecord(robert, 0.6).
    historyRecord(john, 0.8).
23. alcoholRate(antonio, 2).
    alcoholRate(berry, 2).
    alcoholRate(charlie, 2).
    alcoholRate(peter, 2).
    alcoholRate(robert, 3).
    alcoholRate(john, 10).
24. yearsOfExperience(antonio, 5).
    yearsOfExperience(berry, 5).
    yearsOfExperience(charlie, 5).
    yearsOfExperience(peter, 5).
    yearsOfExperience(robert, 2).
    yearsOfExperience(john, 10).
25. endProlog.

```

Someone is considered as a suspect based on the utility value assigned to him. The police compute the utility value of each driver based on the time of his shift, the history record, the amount of alcohol in his blood and the number of years of experience. We assume that all the drivers are honest and answer all of the questions truthfully. Then it is impossible that somebody who had the day shift is still a suspect. The history record is also important in order to see whether the driver is a good person or not. Further, if he has more experience, it means that he is a good driver and lowers the probability of him being a suspect.

After the computation, the police get the result:

```
suspect_person(john, 0.8),
suspect_person(peter, 0.15),
suspect_person(robert, -0.9)
```

and find that Robert is clearly the best candidate for a suspect.

4.3 A Priori Preference

Once the set of relevant considered abducibles is determined from the program's current knowledge state, all that remains is to determine the active a priori preferences that are relevant for that set. This is merely a query for all preference literals whose heads indicate a preference between two abducibles that belong to the set, and whose body is true in the Well-Founded Model of the current knowledge state. Now, we can define the preferences using probabilistic information.

Example 4.3 (Wet Grass) *Suppose that agent Boby wants to have refreshing in the holiday time. There are two options, going to the beach or going to the cinema. Agent Boby prefers to go to the beach if the probability of raining is quite low (less than 40%). Agent Boby also does not want to go to the cinema if the movies are boring. What should agent Boby decide for his refreshing time given the knowledge of weather forecast and the information that today the grass is wet?*

```
1. falsum ← not refreshing.
2. refreshing ← beach.
   refreshing ← cinema.
3. expect(beach).
   expect(cinema).
4. beach ← consider(beach).
5. cinema ← consider(cinema).
6. beach ≺ cinema ← raining(PR),
   prolog(PR < 0.4).
7. raining(PR) ← wetgrass(X),
   prolog(pr((rain(t) | wetgrass(X)), PR)).
8. wetgrass(t) ←
   observe(prog, grass_condition,
   today, true).
9. wetgrass(f) ←
   observe(prog, grass_condition,
   today, false).
10. observe(prog, grass_condition, Q, R) ←
   oracle,
   prolog(oracleQuery(grass_condition(Q), R)).
11. expect_not(cinema) ← boring_movie.
12. boring_movie ←
   observe(prog, movie_reference,
   today, true).
13. observe(prog, movie_reference, Q, R) ←
   oracle,
   prolog(oracleQuery(boring_movie(Q),
   R)).
14. beginPr.
15. bool = {t, f}.
16. cloudy : bool.
17. rain : bool.
18. sprinkler : bool.
19. wetgrass : bool.
20. random(rc, cloudy, full).
21. random(rr, rain, full).
22. random(rs, sprinkler, full).
23. random(rw, wetgrass, full).
24. pa(rc, cloudy(t), d_(1, 2)).
25. pa(rc, cloudy(f), d_(1, 2)).
26. pa(rs, sprinkler(t), d_(1, 2)) :-
   cloudy(f).
27. pa(rs, sprinkler(t), d_(1, 10)) :-
   cloudy(t).
28. pa(rr, rain(t), d_(2, 10)) :-
   cloudy(f).
29. pa(rr, rain(t), d_(8, 10)) :-
   cloudy(t).
30. pa(rw, wetgrass(t), d_(0, 1)) :-
   sprinkler(f), rain(f).
31. pa(rw, wetgrass(t), d_(9, 10)) :-
   sprinkler(t), rain(f).
32. pa(rw, wetgrass(t), d_(9, 10)) :-
   sprinkler(f), rain(t).
33. pa(rw, wetgrass(t), d_(99, 100)) :-
   sprinkler(t), rain(t).
34. endPr.
```

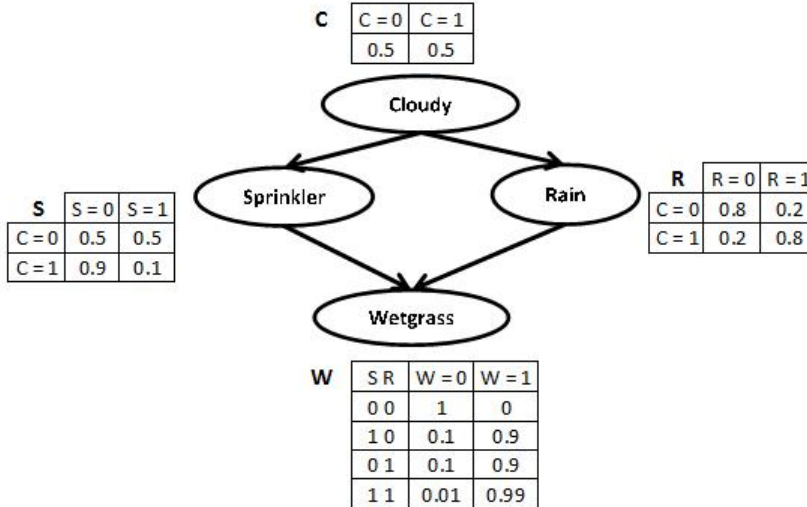


Figure 2: Bayes Nets for Wetgrass Condition

Agent Boby’s preference is based on the value of probability raining given the condition of the grass (lines 6-7). ACORDA calls the oracle to acquire the condition of grass. Later, P-log computes the conditional probability of $rain(t)$ given $wetgrass(X)$. All lines between `beginPr` and `endPr` are reserved for P-log code. The Bayes Nets for this problem is represented in Figure 2 and coded on lines 15-33.

4.4 A Posteriori Preference

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 can not 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.

A given abducible can be defeated in any one of two cases: either by satisfaction of an `expect_not/1` clause 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 that is described later on. Later, ACORDA selects amongst them based on the preference function defined. This steps 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) :-`
`%preference function to select the model.`

Example 4.4 (Tea 3) (*Continuation of Example 3.5*) 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?

age when the utility probability value is taken into account?

```

14. beginProlog.
15.   select(M, Mnew) :-
        utilityRate(tea, 0.8),
        utilityRate(coffee, 0.7),
        select1(M, M2),
        select2(M2, 0, [], Mnew).
16.   select1([], []).
17.   select1([X|Xs], [Y|Ys]) :-
        addUtilityValue(X, Y),
        select1(Xs, Ys).
16.   select2([], -, M, M).
17.   select2([M|Ms], Acc, OldM, NewM) :-
        member(utilityModel(U), M),
        U > Acc → select2(Ms, U, M, NewM);
        select2(Ms, Acc, OldM, NewM).
18.   addUtilityValue([X],
        [utilityModel(UM)|[X]]) :-
        holds utilityRate(X, R),
        pr(available(X), P), UM is R * P.
19.   addUtilityValue(-, []).
20. 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 14-20. 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.

4.5 Multiple-step Prospction

Each cycle ends with the commitment of the agent to an abductive solution that satisfies the current active goals. Sometimes we also want to look ahead several steps based on previously abduced variables in combination with the probability and utility functions. When looking towards the future, an agent is confronted with several scenarios. Later when more information is available (possibly given by an oracle), the future prediction can be repeated.

Example 4.5 (Blue Cab Problem) *A cab was involved in a hit-and-run accident at night. Two cab companies, the Green and the Blue, operate in the city. Imagine you are given the following information: 85% of the cabs in the city are Green and 15% are Blue. A witness identified the cab as a Blue cab. The court tested his ability to identify cabs under the appropriate visibility conditions. When presented with a sample of cabs (half of which were Blue and half of which were Green), the witness made correct identifications in 80% of the cases and erred in 20% of the cases. What is the probability that the cab involved in this accident was Blue?*

1st : find the cab suspect.

There is an accident, we should find the suspect.

```

accident.
false ← accident,
        not find_cab_company_involved.
find_cab_company_involved ←
        find_cab_company_involved(-, -).
find_cab_company_involved(X, P) ←
        consider(suspect(X, P)).
expect(suspect(X, P)) ← cab(X),
        prolog(pr(cab(X) ' | ' witness(blue), P)).
cab(green).
cab(blue).
constrain(2, [suspect(blue, PB),
        suspect(green, PG)], 2) ←
        prolog(pr(cab(blue) ' | ' witness(blue),
        PB)),
        prolog(pr(cab(green) ' | ' witness(blue),
        PG)).
normal ← not check_witness_reliability.
beginPr.
color = {green, blue}.
witness : color.
random(r1, witness, full).
cab : color.

```

```

random(r2, cab, full).
pa(r2, cab(blue), d_(15, 100)) :-
    night(f).
pa(r2, cab(green), d_(85, 100)) :-
    night(f).
pa(r1, witness(blue), d_(80, 100)) :-
    cab(blue), normal.
pa(r1, witness(blue), d_(20, 100)) :-
    cab(green), normal.
night(t) :- holds night.
night(f) :- holds not_night.
normal :- holds normal.
endPr.

```

2nd : Judge the guilty.

From the suspect which one is the guilty?

```

guilty(X) ←
    find_cab_company_involved(X, P),
    prolog(P > 0.5).
guilty ← guilty(_).
falsum ← accident, not observe(guilty).

```

3rd : Defense.

The defendant claims that the accident happened in the night. The number each cabs are different from the number in the day. For Green company, the cabs are reduced one fifth in the night and for the Blue company, the number is reduced into one third.

```
not_night ← not night.
```

beginPr.

```

pa(r2, cab(blue), d_(X, 100)) :-
    night(t), compute(blue, X).
pa(r2, cab(green), d_(X, 100)) :-
    night(t), compute(green, X).
compute(X, Val) :-
    N1 is 85/5 + 15/3,
    (X = blue →
    Val is 5/N1*100; Val is 17/N1*100).
endPr.

```

4th : Prosecutor.

The prosecutor said that the reliability of witness in the night is not good.

beginPr.

```

pa(r1, witness(blue), d_(60, 100)) :-
    cab(blue), night(t),
    check_witness_reliability.
pa(r1, witness(blue), d_(40, 100)) :-
    cab(green), night(t),
    check_witness_reliability.
check_witness_reliability :-
    holds check_witness_reliability.
endPr.

```

5th : Police.

Police is trying to find the suspect.

```

catch_person ← suspect_person(P, U).
expect(suspect(P)) ← driver(P, C),
    find_cab_company_involved(C, PR) ,
    prolog(PR > 0.5).
suspect_person(P, U) ←
    consider(suspect(P)),
    prolog(utilityValue(P, U)),
    prolog(U > 0).
person(P) ← driver(P, _).
driver(antonio, blue).
driver(berry, blue).
driver(charlie, blue).
driver(peter, green).
driver(robert, green).
driver(john, green).
shift(antonio, day).
shift(berry, night).
shift(charlie, day).
shift(peter, night).
shift(robert, night).
shift(john, day).
beginProlog.
utilityValue(Person, 1) :-
    holds shift(Person, night).
utilityValue(Person, 0) :-
    holds shift(Person, day).
endProlog.

```

6th : Interrogation.

The police is asking suspects' alibi.

```

find_alibi ← have_alibi(suspect(P)).
have_alibi(person(P)) ←
    observe(prog, alibi_oracle(P),
    street_abc, true).
observe(prog, alibi_oracle(P), Q, R) ←
    oracle,
    prolog(oracleQuery(was_not_at(P, Q),
    R)).

```

This case can be simulated in below.

Models for step 1:

```

[suspect(blue, 0.4138), suspect(green, 0.5862)]
Continue?(yes/no)yes.
Update knowledge?(yes/no)yes.
falsum ← accident, not observe(guilty).
finish.

```

Models for step 2:

```

[suspect(blue, 0.4138), suspect(green, 0.5862),
    guilty(green)]
Continue?(yes/no)yes.
Update knowledge?(yes/no)yes.
night.

```

```

finish.

Models for step 3:
[suspect(blue, 0.5405), suspect(green, 0.4595),
 guilty(blue)]
Continue?(yes/no)yes.
Update knowledge?(yes/no)yes.
  check_witness_reliability.
  finish.

Models for step 4:
[suspect(blue, 0.3061), suspect(green, 0.6939),
 guilty(green)]
Continue?(yes/no)yes.
Update knowledge?(yes/no)yes.
  falsum <- not catch_person.
  finish.

Models for step 5:
[suspect(peter), suspect(robert),
 suspect(blue, 0.3061), suspect(green, 0.6939),
 guilty(green)],
[suspect(peter), suspect(blue, 0.3061),
 suspect(green, 0.6939), guilty(green)],
[suspect(robert), suspect(blue, 0.3061),
 suspect(green, 0.6939), guilty(green)]
Continue?(yes/no)yes.
Update knowledge?(yes/no)yes.
  expect_not(suspect(P)) <- have_alibi(person(P)).
  finish.

Step: 6
Confirm observation: was_not_at(john, street_abc)
(true or false)? false.
Confirm observation: was_not_at(robert, street_abc)
(true or false)? false.
Confirm observation: was_not_at(peter, street_abc)
(true or false)? true.
Models for step 6:
[suspect(robert), suspect(blue, 0.3061),
 suspect(green, 0.6939), guilty(green)]
Continue?(yes/no)no.

```

5 Example: Risk Analysis

The economics of risk [8] 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 analyze 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 can not 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 5.1 (Mamma Mia) *The owner of the Restaurant “Mamma Mia” 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 “Mamma Mia”? 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,
   prolog(pr(menu(new),  $PN$ )),
   prolog(pr(menu(original),  $PO$ )),
   prolog( $PN > PO$ ).
5. decide(not_launch) ◁
   decide(launch_new_menu) ←
   not_take_risk,
   prolog(pr(menu(original),  $PO$ )),
   prolog(pr(menu(new),  $PN$ )),
   prolog( $PO > PN$ ).
6. not_take_risk ← not take_risk.
7. falsum ← not decision.
8. beginPr.
9.   age = {teenager, adult}.
10.  offer = {original, new}.
11.  customer : age.
12.  random( $rc$ , customer, full).
13.  pa( $rc$ , customer(teenager),  $d_-(60,100)$ ).
14.  menu : offer.
15.  random( $ro$ , menu, full).
16.  pa( $ro$ , menu(new),  $d_-(75,100)$ ) :-
      customer(teenager).
17.  pa( $ro$ , menu(original),  $d_-(80,100)$ ) :-
      customer(adult).
18. endPr.
19. beginProlog.
20. :- import member/2, length/2
      from basics.
21. select( $M$ ,  $Mnew$ ) :-
      select1( $M$ ,  $Mnew$ ),
      select2( $-, 0, [], -$ ).
22. select1( $[], []$ ).
23. select1( $[X|Xs]$ ,  $[Y|Ys]$ ) :-
      addUtilityValue( $X, Y$ ),
      select1( $Xs, Ys$ ).
24. select2( $[], -, M, M$ ).
25. select2( $[M|Ms]$ ,  $Acc, OldM, NewM$ ):-
      member(utilityModel( $U$ ),  $M$ ),
       $U > Acc \rightarrow$  select2( $Ms, U, M, NewM$ );
      select2( $Ms, Acc, OldM, NewM$ ).
26. addUtilityValue( [decide( $X$ )],
                    [utilityModel( $EU$ ),
                     expectedProfit( $Profit$ )])
```

```

                decide( $X$ )] :-
                expectedProfit( $X$ ,  $Profit$ ),
                expectedUtility( $X$ ,  $EU$ ).
27. expectedProfit( $Action$ ,  $P$ ) :-
    return( $Action$ ,  $R$ ), cost( $Action$ ,  $C$ ),
     $P$  is  $R - C$ .
28. expectedUtility( $Action$ ,  $EU$ ) :-
    pr(menu(new),  $PrN$ ),
    expectedProfit( $Action$ ,  $Pf$ ),
     $EU$  is ( $2 * Pf * PrN - 0.01 * Pf * Pf * PrN$ ).
29. return(launch_new_menu, 5).
    return(not_launch, 0).
30. cost(launch_new_menu, 2).
    cost(not_launch, 0).
31. endProlog.
```

In Example 5.1, 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 probability of new menu choices is 0.42 and 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:

$$M_1 = \{\text{utilityModel}(3.1323), \text{expectedProfit}(3), \text{decide}(\text{launch_new_menu})\},$$

$$M_2 = \{\text{utilityModel}(0.0000), \text{expectedProfit}(0), \text{decide}(\text{not_launch})\}$$

Based on the computation, it is better if he launches a new menu even though it is risky.

6 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 can not 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 scenarios, 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.

When looking ahead a number of steps into the future, the agent is confronted with the problem of having several different possible courses of evolution. It needs to be able to prefer amongst them to determine the best courses of evolution from its present state (from any state in general). The (local) preferences, such as the a priori and a posteriori ones presented above are not appropriate enough anymore. The agent should be able to prefer amongst evolutions by their available historical information as well as by quantitatively or qualitatively evaluating their consequences.

Sato proposes PRISM (PRogramming In Statistical Modeling) for logic programs with distribution semantics [26]. P-log and PRISM share a substantial number of common features. Both are declarative languages capable of representing and reasoning with logical and probabilistic knowledge. In both cases logical part of the language is rooted in logic programming. There are also substantial differences. PRISM allows infinite possible worlds, has the ability for statistical parameters learning embedded in its

inference mechanism, but limits its logical power to Horn logic programs. We can use PRISM ideas to expand the semantics of P-log to allow infinite possible worlds.

In Section 5 we show the analysis of risk behaviour under general risk preferences under the expected utility model. However, applying this approach to decision-making under uncertainty requires having good information about the measurement of the probability distribution of x and the risk preferences of the decision-maker as represented by utility function $U(x)$. It is easier to obtain sample information about the probability distribution of x than about individual risk preferences. It is possible to conduct risk analysis without precise information about risk preferences using stochastic dominance. Stochastic dominance provides a framework to rank choices among alternative risky strategies when preferences are not precisely known [28]. It foresees 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.

7 Acknowledgements

We thank Gonalo Lopes from CENTRIA for his help with ACORDA and Han The Anh from CENTRIA for his help with P-log. We thank Weijun Zhu for his help with the original P-log.

References

- [1] J. J. Alferes, A. Brogi, J. A. Leite, and L. M. Pereira. Evolving logic programs. In S. Flesca, S. Greco, N. Leone, and G. Ianni, editors, *Procs. of the 8th European Conf. on Logics in Artificial Intelligence (JELIA'02)*, LNCS 2424, pages 50–61. Springer, 2002.
- [2] H. T. Anh, C. D. P. K. Ramli, and C. V. Damasio. An implementation of extended p-log using xasp. LNCS 5366, pages 739–743. Springer, 2008.

- [3] C. Baral, M. Gelfond, and N. Rushton. Probabilistic reasoning with answer sets. In *LPNMR7*, LNAI 2923, pages 21–33, 2004.
- [4] C. Baral, M. Gelfond, and N. Rushton. Probabilistic reasoning with answer sets. Volume 9, Part 1, pages 57–144, January 2009.
- [5] J. Baron. *Thinking and Deciding*. Cambridge University Press, 2000.
- [6] W. H. Calvin. *How Brains Think: Evolving Intelligence, Then And Now*. Basic Books, 1996.
- [7] L. Castro, T. Swift, and D. S. Warren. Xasp: Answer set programming with xsb and smodels. Accessed at <http://xsb.sourceforge.net/packages/xasp.pdf>.
- [8] J. P. Chavas. *Risk Analysis in Theory and Practice*. Academic Press, 2004.
- [9] A. Damásio. *Looking for Spinoza: Joy, Sorrow and the Feeling Brain*. Harcourt, 2003.
- [10] P. Dell’Acqua and L. M. Pereira. Preferential theory revision (extended version). 5(4), pages 586 – 601. Elsevier.
- [11] D. C. Dennett. *Sweet Dreams: Philosophical Obstacles to a Science of Consciousness*. The MIT Press, 2005.
- [12] M. Donald. *A Mind So Rare: The Evolution of Human Consciousness*. W. W. Norton & Company, London, 2001.
- [13] M. Gelfond, N. Rushton, and W. Zhu. Combining logical and probabilistic reasoning. In *AAAI Spring Symposium*, 2006.
- [14] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth Bowen, editors, *Proceedings of the Fifth International Conference on Logic Programming*, pages 1070–1080, Cambridge, Massachusetts, 1988. The MIT Press.
- [15] D. Hume. *An Enquiry Concerning Human Understanding: A Critical Edition*. Oxford Philosophical Texts, 1748.
- [16] D. Lewis. Causation and postscripts to “causation”. *Philosophical Papers, Vol. II*. Oxford: Oxford University Press, pages 172–213, 1986.
- [17] J. Mackie. *The Cement of the Universe*. Oxford: Clarendon Press, 1974.
- [18] D. V. McDermott. *Mind and Mechanism*. The MIT Press, 2001.
- [19] J. Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2000.
- [20] L. M. Pereira, P. Dell’Acqua, and G. Lopes. On preferring and inspecting abductive models. In *Procs. 11th Intl. Symp. Practical Aspects of Declarative Languages (PADL’09)*, LNCS 5418, pages 1–15. Springer, January 2009.
- [21] L. M. Pereira, P. Dell’Acqua, and A. M. Pinto. Preferring abductive models and inspecting side-effects for decision making (extended version of [20]). In *AI journal*, February 2009.
- [22] L. M. Pereira and G. Lopes. Prospective logic agents. In *AI Procs. 13th Portuguese Intl. Conf. on Artificial Intelligence (EPIA’07)*, LNAI 4784, pages 73–86. Springer, 2007.
- [23] L. M. Pereira and G. Lopes. Prospective logic agents (extended version of [21]). In *International Journal of Reasoning-based Intelligent Systems (IJRIS)*, 2009.
- [24] L. M. Pereira and G. Lopes. Prospective logic agents. *Procs. 13th Portuguese Intl. Conf. on Artificial Intelligence*, pages 73–86, December 2007.
- [25] L. M. Pereira and A. M. Pinto. Revised stable models - a semantics for logic programs. In *12th Portuguese Intl. Conf. on Artificial Intelligence (EPIA’05)*, LNAI 3808, pages 29–42. Springer, 2005.

- [26] T. Sato. A statistical learning method for logic programs with distribution semantics. In *In Proceedings of the 12th International Conference on Logic Programming (ICLP95)*, pages 715–729. MIT Press, 1995.
- [27] H. A. Simon and N. Rescher. Cause and counterfactual. *Philosophy of Science*, Vol. 33, No. 4, pages 323–340, December 1966.
- [28] G. A. Whitmore and M. C. Findlay. *Stochastic Dominance: An Approach to Decision-Making Under Risk*. Lexington Books, D.C. Heath and Co, Lexington, MA, 1978.
- [29] R.A. Wilson and F.C. Keil, editors. *The MIT Encyclopedia of the Cognitive Sciences*. The MIT Press, 1999.