Intention Recognition with Evolution Prospection and Causal Bayes Networks

Luís Moniz Pereira and Han The Anh
lmp@di.fct.unl.pt, h.anh@fct.unl.pt

Centro de Inteligência Artificial (CENTRIA)
Departamento de Informática, Faculdade de Ciências e Tecnologia,
Universidade Nova de Lisboa, 2829-516 Caparica, Portugal

Abstract. We describe a novel approach to tackle intention recognition, by combining dynamically configurable and situation-sensitive Causal Bayes Networks plus plan generation techniques.

Given some situation, such networks enable the recognizing agent to come up with the most likely intentions of the intending agent, i.e. solve one main issue of intention recognition. And, in case of having to make a quick decision, focus on the most important ones. Furthermore, the combination with plan generation provides a significant method to guide the recognition process with respect to hidden actions and unobservable effects, in order to confirm or disconfirm likely intentions. The absence of this articulation is a main drawback of the approaches using Bayes Networks solely, due to the combinatorial problem they encounter.

We explore and exemplify its application, in the Elder Care context, of the ability to perform Intention Recognition and of wielding Evolution Prospection methods to help the Elder achieve its intentions. This is achieved by means of an articulate use of a Causal Bayes Network to heuristically gauge probable general intention – combined with specific generation of plans involving preferences – for checking which such intentions are plausibly being carried out in the specific situation at hand, and suggesting actions to the Elder. The overall approach is formulated within one coherent and general logic programming framework and implemented system. The paper recaps required background and illustrates the approach via an extended application example.

Keywords: Intention recognition, Elder Care, Causal Bayes Networks, Plan generation, Evolution Prospection, Preferences, Logic Programming.

1 Introduction

In many multi-agent systems, the problem of intention recognition appears to be crucial when the agents cooperate or compete to achieve a certain task, especially when the possibility of communication is limited. For example, in heterogeneous agent systems it is likely that agents speak different languages, have different designs or different levels of intelligence; hence, intention recognition may be

the only way the agents understand each other so as to secure a successful cooperation. Moreover, when competing, the agents even often attempt to hide their real intentions and make others believe in some pretense ones. Intention recognition in this setting becomes undoubtedly crucial for agents, in order to prepare themselves from potential hostile behaviors from others.

Needless to say, the recognized intentions provide the recognizing agent with valuable information in dealing with other agents, whether they cooperate or compete with each other. But how this information can be valuable for the recognizing agent? In this work, besides the problem of intention recognition, we attempt to address that issue using our implemented Evolution Prospection Agent system [2,3].

Recently, there have been many works addressing the problem of intention recognition as well as its applications in a variety of fields. As in Heinze's doctoral thesis [15], intention recognition is defined, in general terms, as the process of becoming aware of the intention of another agent and, more technically, as the problem of inferring an agent's intention through its actions and their effects on the environment. According to this definition, one approach to tackle intention recognition is by reducing it to plan recognition, i.e. the problem of generating plans achieving the intentions and choosing the ones that match the observed actions and their effects in the environment of the intending agent. This has been the main stream so far [15,18].

One of the core issues of that approach is that of finding an initial set of possible intentions (of the intending agent) that the plan generator is going to tackle, and which must be imagined by the recognizing agent. Undoubtedly, this set should depend on the situation at hand, since generating plans for all intentions one agent could have, for whatever situation he might be in, is unrealistic if not impossible.

In this work, we use an approach to solve this problem employing so-called *situation-sensitive Causal Bayes Networks* (CBN) - That is, CBNs [25] that change according to the situation under consideration, itself subject to ongoing change as a result of actions. Therefore, in some given situation, a CBN can be configured dynamically, to compute the likelihood of intentions and filter out the much less likely ones. The plan generator (or plan library) thus only needs, at the start, to deal with the remaining more relevant because more probable or credible intentions, rather than all conceivable intentions. One of the important advantages of our approach is that, on the basis of the information provided by the CBN the recognizing agent can see which intentions are more likely and worth addressing, so, in case of having to make a quick decision, it can focus on the most relevant ones first. CBNs, in our work, are represented in P-log [5,8,6], a declarative language that combines logical and probabilistic reasoning, and uses Answer Set Programming (ASP) as its logical and CBNs as its probabilistic foundations. Given a CBN, its situation-sensitive version is constructed by attaching to it a logical component to dynamically compute situation specific probabilistic information, which is forthwith inserted into the P-log program representing that CBN. The computation is dynamic in the sense

that there is a process of inter-feedback between the logical component and the CBN, i.e. the result from the updated CBN is also given back to the logical component, and that might give rise to further updating, etc.

In addition, one more advantage of our approach, in comparison with the stream of those using solely BNs [16,17] is that these just use the available information for constructing CBNs. For complicated tasks, e.g. in recognizing hidden intentions, not all information is observable. Whereas CBNs are appropriate for coding general average information, they quickly bog down in detail when aspiring to code multitudes of specific situations and their conditional probability distributions. The approach of combining CBNs with plan generation provides a way to guide the recognition process: which actions (or their effects) should be checked whether they were (hiddenly) executed by the intending agent. So, plan recognition ties the average statistical information with the situation particulars, and obtains specific situational information that can be fed into the CBN. In practice, one can make use of any plan generators or plan libraries available. For integration's sake, we can use the ASP based conditional planner called ASCP [20] from XSB Prolog using the XASP package [9,32] for interfacing with Smodels [30] – an answer set solver – or, alternatively, rely on plan libraries so obtained.

The next step, that of taking advantage of the recognized intention gleaned from the previous stage, is implemented using our Evolution Prospection Agent (EPA) system [2,3]. The latter allows an agent to be able to look ahead, prospectively, into its hypothetical futures, in order to determine the best courses of evolution that satisfy its goals, and thence to prefer amongst those futures. These courses of evolution can be provided to the intending agent as suggestions to achieve its intention (in cooperating settings) or else as a guide to prevent that agent from achieving it (in hostile settings).

In EPA system, *a priori* and *a posteriori* preferences, embedded in the knowledge representation theory, are used for preferring amongst hypothetical futures. The *a priori* ones are employed to produce the most interesting or relevant conjectures about possible future states, while the *a posteriori* ones allow the agent to actually make a choice based on the imagined consequences in each scenario. In addition, different kinds of evolution-level preferences enable agents to attempt long-term goals, based on the historical information as well as quantitative and qualitative a posteriori evaluation of the possible evolutions

In the sequel we describe the intention recognition and evolution prospection systems, showing an extended example for illustration. Then, Elder Care – a real world application domain, is addressed by the combination of the two systems. The paper finishes with Conclusions and Future directions ...

2 Intention Recognition

2.1 Causal Bayes Networks

We briefly recall Causal Bayes Networks (CBN) here for convenience in order to help understand their use for intention recognition and their realization in P-log. Firstly, let us recall some preliminary definitions.

Definition 1 (Directed Acyclic Graph). *A directed acyclic graph, also called a dag, is a directed graph with no directed cycles; that is, for any node v, there is no nonempty directed path that starts and ends on v.*

Definition 2. *Let G be a dag that represents causal relations between its nodes. For two nodes A and B of G, if there is an edge from A to B (i.e. A is a direct cause of B), A is called a parent of B, and B is a child of A. The set of parent nodes of a node A is denoted by **parents(A)**. Ancestor nodes of A are parents of A or parents of some ancestor nodes of A. If node A has no parents ($parents(A) = \emptyset$), it is called a **top node**. If A has no child, it is called a **bottom node**. The nodes which are neither top nor bottom are said **intermediate**. If the value of a node is observed, the node is said to be an **evidence node**.*

Definition 3 (Causally Sufficient Set [23]). *Let V be a set of variables with causal relations represented by a dag. V is said to be causally sufficient if and only if for any $Y, Z \in V$ with $Y \neq Z$ and X is a common cause of Y and Z, then $X \in V$. That is to say, V is causally sufficient if for any two distinct variables in V, all their common causes also belong to V.*

Definition 4 (Causal Markov Assumption - CMA [23]). *Let X be any variable in a causally sufficient set S of variables or features whose causal relations are represented by a dag G, and let P be the set of all variables in S that are direct causes of X (i.e. parents of X in G). Let Y be any subset of S such that no variable in Y is a direct or indirect effect of X (i.e., there is no directed path in G from X to any member of Y). Then X is independent (in probability) of Y conditional on P.*

The CMA implies that the joint probability of any set of values of a causally sufficient set can be factored into a product of conditional probabilities of the value of each variable on its parents. More details and a number of examples can be found in [23].

Definition 5 (Bayes Networks). *A Bayes Network is a pair consisting of a dag whose nodes represent variables and missing edges encode conditional independencies between the variables, and an associated probability distribution satisfying the Causal Markov Assumption.*

If the dag of a BN is intended to represent causal relations and its associated probability distribution is intended to represent those that result from the represented mechanism, then the BN is said to be causal. To do so, besides CMA, the associated probability distribution needs to satisfy an additional condition, as more formally shown in the following definition

Definition 6 (Causal Bayes Network). *A Bayes Network is causal if its associated probability distribution satisfies the condition specifying that if a node X of its dag is actively caused to be in a given state x (an operation written as do(x), e.g. in P-log syntax), then the probability density function changes to the one of the network obtained by cutting the links from X's parents to X, and setting X to the caused value x [25].*

With this condition being satisfied, one can predict the impact of external interventions from data obtained prior to intervention.

In a BN, associated with each intermediate node of its dag is a specification of the distribution of its variable, say A, conditioned on its parents in the graph, i.e. $P(A|parents(A))$ is specified. For a top node, the unconditional distribution of the variable is specified. These distributions are called Conditional Probability Distribution (CPD) of the BN.

Suppose nodes of the dag form a causally sufficient set, i.e. no common causes of any two nodes are omitted, then implied by CMA [23], the joint distribution of all node values of a causally sufficient can be determined as the product of conditional probabilities of the value of each node on its parents

$$P(X_1, ..., X_N) = \prod_{i=1}^{N} P(X_i|parents(X_i))$$

where $V = \{X_i|1 \leq i \leq N\}$ is the set of nodes of the dag.

Suppose there is a set of evidence nodes in the dag, say $O = \{O_1, ..., O_m\} \subset V$. We can determine the conditional probability of a variable X given the observed value of evidence nodes by using the conditional probability formula

$$P(X|O) = \frac{P(X, O)}{P(O)} = \frac{P(X, O_1, ..., O_m)}{P(O_1, ..., O_m)} \tag{1}$$

where the numerator and denominator are computed by summing the joint probabilities over all absent variables w.r.t. V as follows

$$P(X = x, O = o) = \sum_{av \in ASG(AV_1)} P(X = x, O = o, AV_1 = av)$$
$$P(O = o) = \sum_{av \in ASG(AV_2)} P(O = o, AV_2 = av)$$

where $o = \{o_1, ..., o_m\}$ with $o_1, ..., o_m$ being the observed values of $O_1, ..., O_m$, respectively; $ASG(Vt)$ denotes the set of all assignments of vector Vt (with components are variables in V); AV_1, AV_2 are vectors components of which are corresponding absent variables, i.e. variables in $V \setminus \{O \cup \{X\}\}$ and $V \setminus O$, respectively.

In short, to define a BN, one needs to specify the structure of the network, its CPD and the prior probability distribution of the top nodes. We will see some examples later, in Figures 3 and 9.

2.2 Intention recognition with Causal Bayesian Networks

The first phase of the intention recognition system is to find out how likely each conceivable intention is, based on current observations such as observed actions of the intending agent or the effects its actions (either actually observed, or missed direct observation) have in the environment. A conceivable intention is the one having causal relations to all current observations. It is carried out by using a CBN with nodes standing for binary random variables that represent causes, intentions, actions and effects. The structure of the CBN and its components such as CPD and probability distribution of top nodes are specified in the sequel.

To begin with, for better understand the structure of CBNs for intention recognition we will present, let us recall the high-level model of intentional behavior described in [15]. In this work, Heinze [15] proposed a tri-level decompositional model of intentional behavior of the intending agent. Intention recognition is the reversal of this process. These levels are:

- **The intentional level** describes the intentions of agent in terms of desires, beliefs, goals, plans and other high-level intentional states. These intentions give rise, in a directly causal way, to activities.
- **The activity level** describes the activities and actions undertaken by the agent. The actions are a direct result of the intentional state of the agent. The activities can be to select plans, adopt tactics, etc.
- **The state level** describes the agent in terms of externally accessible characteristics and effects its actions have in the environment.

According to this description, six basic paths (approaches) can be followed in intention recognition as shown in Figure 1. For example, scheme 1 corresponds to the basic '*sense and infer*' approach, scheme 3 matches the trivial case of communicating intentions, while scheme 6 resembles direct action recognition from state and inferring the recognized action in intention.

We may also think of these six basic paths as all possible cases that can happen when recognizing intentions of an agent, depending on the information available. For example, when not able to observe actions of the intending agent, the recognizing agent must attempt to analyze the accessible states of the environment to figure out which actions might cause those states to the environment. Depending on to which level of intentional behavior the observable information belongs to, an intention recognition system should be able to flexibly employ the right scheme.

Based on this tri-level model with a small modification, we next describe a structure of CBNs that allows computing the likelihood of intentions, given the observable information. A fourth level that describes the causes which might give rise to the considered intentions is added. That level is called *pre-intentional*.

The structure is as follows. Intentions are represented by intermediate nodes whose ancestor nodes represent causes that give rise to those intentions. The causes, as mentioned above, belong to the pre-intentional level of a model whose intentional level contains the intentions. Intuitively, the additional level is introduced, first of all, to help with estimating prior probabilities of the intentions.

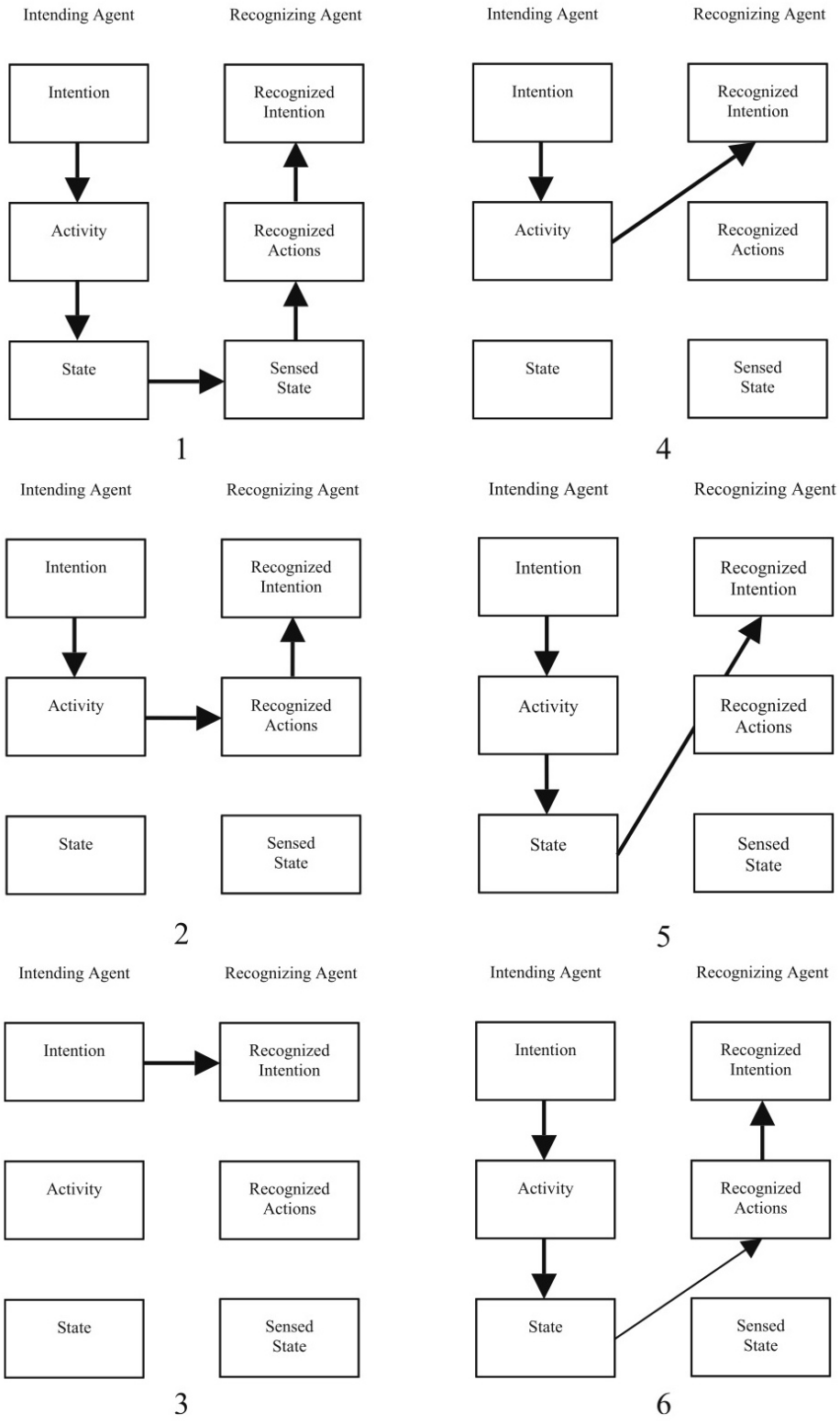

Fig. 1: Six Possible Paths in Intention Recognition [15]

Secondly, it guarantees the causal sufficiency condition of the set of variables represented by the nodes of the dag. However, if these prior probabilities can be specified without considering the causes, intentions are represented by top nodes. Top nodes reflect the problem context or the intending agent's mental state. Note that there might be top nodes which are evidence ones, i.e. being observable. In our CBN for intention recognition, evidence nodes need not to be only the observed actions which result from having some intentions. Later we will see an example (Elder Care, Figure 9) having observed top nodes. Observed

Fig. 2: Fox and Crow

actions are represented as children of the intentions that causally affect them. Observable effects are represented as bottom nodes. They can be children of observed action nodes, of intention nodes, or of some unobserved actions that might cause the observable effects that are added as children of the intention nodes.

The above causal relations (e.g. which causes give rise to an intention, which intentions trigger an action, which actions have an effect) among nodes of the BNs, as well as its CPD and the distribution of the top nodes, are specified by domain experts. However, they are also possible to learn automatically. Finally, by using formula (1) the conditional probabilities of each intention on current observations can be determined, X being an intention and O being the set of current observations.

Example 1 (Fox-Crow). Consider Fox-Crow story - adapted from Aesop's fable (Figure 2). There is a crow, holding a cheese. A fox, being hungry, approaches the crow and praises her, hoping that the crow will sing and the cheese will fall down near him. Unfortunately for the fox, the crow is very intelligent, having the ability of intention recognition.

The Fox's intentions CBN is depicted in the Figure 3. The initial possible intentions of Fox that Crow comes up with are: Food - $i(F)$, Please - $i(P)$ and Territory - $i(T)$. The facts that might give rise to those intentions are how friendly the Fox is (*Friendly_fox*) and how hungry he is (*Hungry_fox*). Currently, there is only one observation which is: Fox praised Crow ($Praised$).

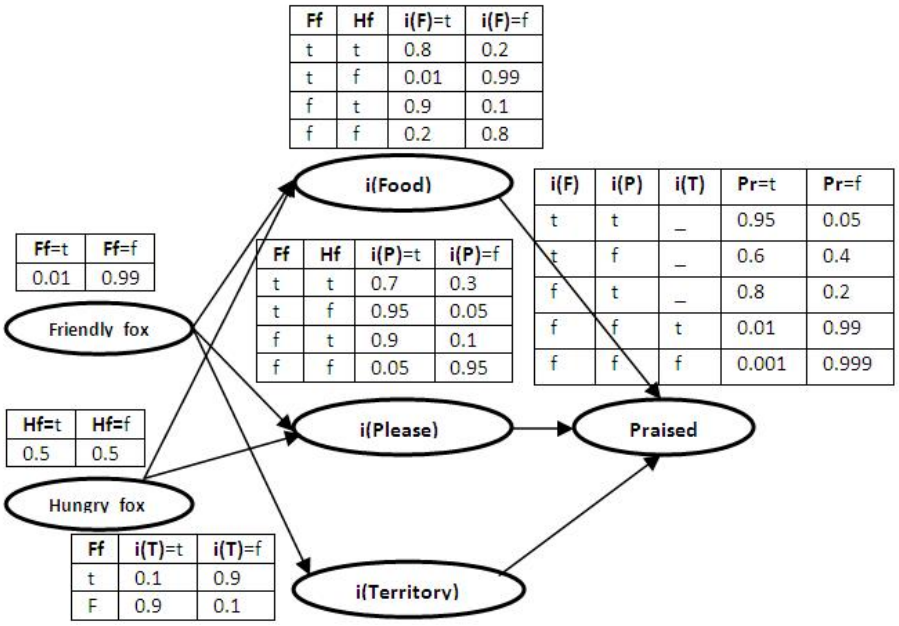

Fig. 3: Fox's Intentions CBN

2.3 P-log

The computation in CBNs is automated using P-log, a declarative language that combines logical and probabilistic reasoning, and ASP as its logical and CBNs as its probabilistic foundations. We recap it here for self-containment, to the extent we use it.

The original P-log [5,8] uses 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 a variety of combinatorial problems, its non-relevance property [9]

makes the P-log system sometimes computationally redundant. Newer developments of P-log [6] use the XASP package of XSB Prolog [32] for interfacing with Smodels [30] – an answer set solver. The power of ASP allows the representation of both classical and default negation in P-log easily. Moreover, the new P-log uses XSB as the underlying processing platform, allowing arbitrary Prolog code for recursive definitions. Consequently, it allows more expressive queries not supported in the original version, such as meta queries (probabilistic built-in predicates can be used as usual XSB predicates, thus allowing full power of probabilistic reasoning in XSB) and queries in the form of any XSB predicate expression [6]. Moreover, the tabling mechanism of XSB [31] significantly improves the performance of the system.

In general, a P-log program Π consists of the 5 components detailed below: a sorted signature, declarations, a regular part, a set of random selection rules, a probabilistic information part, and a set of observations and actions.

(i) Sorted signature and Declaration The sorted signature Σ of 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 reserved function symbols called attributes. 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. A literal is an atomic statement, p, or its explicit negation, neg_p.

The declaration part of a P-log program can be defined as a collection of sorts and sort declarations of attributes. A sort c can be defined by listing all the elements $c = \{x_1, ..., x_n\}$, specifying the range of values $c = \{L..U\}$ where L and U are the integer lower bound and upper bound of the sort c. Attribute a with domain $c_1 \times ... \times c_n$ and range c_0 is represented as follows:

$$a : c_1 \times ... \times c_n \texttt{ --> } c_0$$

If attribute a has no domain parameter, we simply write $a : c_0$. The range of attribute a is denoted by $range(a)$.

(ii) Regular part This part of a P-log program consists of a collection of rules, facts, and integrity constraints (IC) in the form of denials, formed using literals of Σ. An IC is encoded as a rule with the `false` literal in the head.

(iii) Random Selection Rule This is a rule for attribute a having the form:

$$random(RandomName, a(\bar{t}), DynamicRange) \texttt{ :- } Body$$

This means that the attribute instance $a(\bar{t})$ is random if the conditions in $Body$ are satisfied. The $DynamicRange$, not used in the particular examples in the sequel, allows to restrict the default range for random attributes. The $RandomName$ is a syntactic mechanism used to link random attributes to the corresponding probabilities. If there is no precondition, we simply put $true$ in the body. A constant $full$ can be used in $DynamicRange$ to signal that the dynamic domain is equal to $range(a)$.

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

$$pa(RandomName, a(\bar{t}, y), d\_(A, B)) \texttt{:-}\ Body.$$

meaning that if the *Body* were true, and the value of $a(\bar{t})$ were selected by a rule named *RandomName*, then *Body* would cause $a(\bar{t}) = y$ with probability $\frac{A}{B}$.

(v) Observations and Actions These 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. of random attributes and attributes dependent on them. The statement $do(a(t, y))$ indicates that $a(t) = y$ is enforced true as the result of a deliberate action, not an observation.

2.4 Recognizing Fox's intentions - An Example

Example 2 (Fox-Crow – Example 1 cont'd). The Fox's intentions CBN can be coded with the P-log program in Figure 4.

Two sorts *bool* and $fox\_intentions$, in order to represent boolean values and set of Fox's intentions, are declared in part 1. Part 2 is the declaration of four attributes $hungry\_fox$, $friendly\_fox$, *praised* and i which state the first three attributes have no domain parameter and get boolean values, and the last one maps each Fox's intention to a boolean value. The random selection rules in part 3 declare that these four attributes are randomly distributed in their ranges. The distributions of the top nodes ($hungry\_fox$, $friendly\_fox$) and the CPD corresponding to the CBN in Figure 3 are given in part 4 and parts 5-8, respectively, using the probabilistic information pa-rules. For example, in part 4 the first rule says that fox is hungry with probability 1/2 and the second rule says he is friendly with probability 1/100. The first rule in part 5 states that if Fox is friendly and hungry, the probability of him having intention Food is 8/10.

Note that the probability of an atom $a(\bar{t}, y)$ will be directly assigned if the corresponding $pa/3$ atom is in the head of some *pa-rule* with a true body. To define probabilities of the remaining atoms we assume that by default, all values of a given attribute which are not assigned a probability are equally likely. For example, first rule in part 4 implies that fox is not hungry with probability 1/2. And, actually, we can remove that rule without changing the probabilistic information since, in that case, the probability of fox being hungry and of not being so are both defined by default, thus, equal to 1/2.

The probabilities of Fox having intention Food, Territory and Please given the observation that Fox praised Crow can be found in P-log with following queries, respectively,

? – pr(i(food, t) | obs(praised(t)), V_1). The answer is: $V_1 = 0.9317$.
? – pr(i(territory, t) | obs(praised(t)), V_2). The answer is: $V_2 = 0.8836$.
? – pr(i(please, t) | obs(praised(t)), V_3). The answer is: $V_3 = 0.0900$.

```
1. bool = {t,f}.   fox_intentions = {food,please,territory}.
2. hungry_fox : bool.                 friendly_fox : bool.
   i : fox_intentions --> bool.       praised : bool.
3. random(rh, hungry_fox, full).      random(rf, friendly_fox, full).
   random(ri, i(I), full).            random(rp, praised, full).
4. pa(rh,hungry_fox(t),d_(1,2)).      pa(rf,friendly_fox(t),d_(1,100)).
5. pa(ri(food),i(food,t),d_(8,10))   :-friendly_fox(t),hungry_fox(t).
   pa(ri(food),i(food,t),d_(9,10))   :-friendly_fox(f),hungry_fox(t).
   pa(ri(food),i(food,t),d_(0.1,10)) :-friendly_fox(t),hungry_fox(f).
   pa(ri(food),i(food,t),d_(2,10))   :-friendly_fox(f),hungry_fox(f).
6. pa(ri(please),i(please,t),d_(7,10))   :-friendly_fox(t),hungry_fox(t).
   pa(ri(please),i(please,t),d_(1,100))  :-friendly_fox(f),hungry_fox(t).
   pa(ri(please),i(please,t),d_(95,100)) :-friendly_fox(t),hungry_fox(f).
   pa(ri(please),i(please,t),d_(5,100))  :-friendly_fox(f),hungry_fox(f).
7. pa(ri(territory),i(territory,t),d_(1,10)) :-friendly_fox(t).
   pa(ri(territory),i(territory,t),d_(9,10)) :-friendly_fox(f).
8. pa(rp, praised(t),d_(95,100)) :-i(food,t),i(please,t).
   pa(rp, praised(t),d_(6,10))   :-i(food,t),i(please,f).
   pa(rp, praised(t),d_(8,10))   :-i(food,f),i(please,t).
   pa(rp, praised(t),d_(1,100))  :-i(food,f),i(please,f),i(territory,t).
   pa(rp, praised(t),d_(1,1000)) :-i(food,f),i(please,f),i(territory,f).
```

Fig. 4: Fox's intentions CBN

From the result we can say that Fox is very unlikely to have the intention Please, i.e. to make the Crow pleased since its likelihood is very much less than the others. Thus, the next step of Crow's intention recognition is to generate conceivable plans that might corroborate the two remaining intentions. The one with greater likelihood will be discovered first.

2.5 Situation-sensitive CBNs.

Undoubtedly, CBNs should be situation-sensitive since using a general CBN for all specific situations (instances) of a problem domain is unrealistic and most likely imprecise. For example, in the Fox-Crow scenario the probabilistic information in Crow's CBN about the Fox's intention of getting Crow's territory very much depends on what kind of territories the Crow occupies. However, consulting the domain expert to manually change the CBN w.r.t. each situation is also very costly. We here provide a way to construct situation-sensitive CBNs, i.e. ones that change according to the given situation. It uses Logic Programming (LP) techniques to compute situation specific probabilistic information which is then updated into a CBN general for the problem domain.

The LP techniques can be deduction with top-down procedure (Prolog) (to deduce situation-specific probabilistic information) or abduction (to abduce probabilistic information needed to explain observations representing the given situation). However, we do not exclude various other types of reasoning, e.g. in-

cluding integrity constraint satisfaction, abduction, contradiction removal, preferences, or inductive learning, whose results can be compiled (in part) into an evolving CBN.

The issue of how to update a CBN with new probabilistic information can take advantage of the advance in LP semantics for evolving programs with updates [27,28,29]. However, in this work we employ a simpler way, demonstrated in the following example.

Example 3 (Fox-Crow (cont'd)). Suppose the fixed general CBN is the one given in Figure 4. The Prolog program contains the following two rules for updating the probabilistic information in part 7 of the CBN:

```
pa_rule(pa(ri(territory),i(territory,t),d_(0,100)),[friendly_fox(t)])
      :- territory(tree).
pa_rule(pa(ri(territory),i(territory,t),d_(1,100)),[friendly_fox(f)])
      :- territory(tree).
```

Given a P-log probabilistic information pa-rule, then the corresponding so-called situation-sensitive $pa\_rule/2$ predicate takes the head and body of the pa-rule as its first and second arguments, respectively. A situation is given, in this work, by asserted facts representing it. In order to find the probabilistic information specific for the given situation, we simply use the XSB built-in $findall/3$ predicate to find all true $pa\_rule/3$ literals.

In the story the Crow's territory is a tree, thus the fact $territory(tree)$ is asserted. Hence, the following two $pa\_rule/3$ literals are true

```
pa_rule(pa(ri(territory),i(territory,t),d_(0,100)),[friendly_fox(t)])
pa_rule(pa(ri(territory),i(territory,t),d_(1,100)),[friendly_fox(f)])
```

The CBN is updated by replacing the two pa-rules in part 7 of the CBN with the corresponding two rules

```
pa(ri(territory),i(territory,t),d_(0,100)) :- friendly_fox(t)
pa(ri(territory),i(territory,t),d_(1,100)) :- friendly_fox(f)
```

This change can be easily made at the preprocessing stage of the implementation of P-log(XSB) (more details about the system implementation can be found in [6]).

In this updated CBN the likelihood of the intentions $i(food, t)$, $i(territory, t)$, $i(please, t)$ are: $V_1 = 0.9407$; $V_2 = 0.0099$; $V_3 = 0.0908$, respectively. Thus, much likely, the only surviving intention is $food$.

2.6 Plan Generation

The second phase of the intention recognition system is to generate conceivable plans that can achieve the most likely intentions surviving after the first phase. Any appropriate planners, though those implemented in ASP and/or Prolog are preferable for integration's sake, might be used for this task, e.g. $DLV^{\mathcal{K}}$ – a declarative, logic-based planning system built on top of the DLV [19] and ASCP – an ASP based conditional planner [20].

In our system, plan generation is carried out by a new implementation of ASCP in XSB Prolog using XASP package [21]. It has the same syntax and uses the same transformation to ASP as in the original version. It might have better performance because of the relevance property and tabling mechanism in XSB, but we will not discuss that here. Next we briefly recall the syntax of ASCP necessary to represent the example being considered. Semantics and the transformation to ASP can be found in [20].

2.7 Action language $\mathcal{A}_K^c$

ASCP uses A_K^c - a representation action language that extends $\mathcal{A}$ [22] by introducing new types of propositions called *knowledge producing proposition* and *executability condition*, and *static causal laws*.

The alphabet of A_K^c consists of a set of actions $\mathbf{A}$ and a set of fluents $\mathbf{F}$. A *fluent literal* (or *literal* for short) is either a fluent $f \in \mathbf{F}$ or its negation $\neg f$. A fluent formula is a propositional formula φ constructed from the set of literals using operators $\wedge, \vee$ and/or $\neg$. To describe an action theory, 5 kinds of propositions used: (1) **initially**(l); (2) **executable**(a, ψ); (3) **causes**(a, l, ϕ); (4) **if**(l, φ); and (5) **determines**(a, θ).

The initial situation is described by a set of propositions (1), called v-propositions. (1) says that l holds in the initial situation. A proposition of form (2) is called executability condition. It says that a is executable in any situation in which ψ holds. A proposition (3), called a dynamic causal law, saying that performing a in a situation in which ϕ holds causes l to hold in the successor situation. A proposition (4), called a static causal law, states that l holds in any situation in which φ holds. A knowledge proposition (5) states that the values of literals in θ, sometimes referred to as sensed-literals, will be known after a is executed.

A *planning problem instance* is a triple $\pi = (D, I, G)$ where D is a set of propositions of types from (2) to (5), called domain description; I is a set of propositions of type (1), dubbed initial situation; and G is a conjunction of fluent literals.

With the presence of sensing actions we need to extend the notion of plans from a sequence of actions so as to allow conditional statements of the form **case-endcase** (which subsumes the **if-then** statement). A conditional plan can be empty, i.e. containing no action, denoted by []; or sequence [a; p] where a is a non-sensing action and p is a conditional plan; or conditional sequence $[a; cases(\{g_j \rightarrow p_j\}_{j=1}^n]$ where a is a sensing action of a proposition (5) with $\theta = \{g1, ..., gn\}$ and p_j's are conditional plans; Nothing else is a conditional plan.

To execute a conditional plan of the form $[a; cases(\{g_j \rightarrow p_j\}_{j=1}^n]$, we first execute a and then evaluate each g_j w.r.t. our current knowledge. If one of the g_j's, say g_k holds, we execute p_k.

ASCP planner works by transforming a given planning problem instance into an ASP program whose answer sets correspond to conditional plans of the problem instance (see [20] for details).

2.7.1 Representation in the action language We now show how the Crow represents Fox's actions language and two problem instances corresponding to the two Fox's intentions, gathered from the CBN: *Food* (not to be hungry) and *Territory* (occupy Crow's tree) in A_K^c. The representation is inspired by the work in [24].

Example 4 (Fox-Crow (cont'd)). The scenarios with intentions of getting food and territory are represented in Figure 5 and 6, respectively. The first problem instance has the conditional plan:

$[praise(fox, crow), \textbf{cases}(\{$
$\quad \mathsf{accepted(crow)} \rightarrow [sing(crow), grab(fox, cheese), eat(fox, cheese)];$
$\quad \mathsf{declined(crow)} \rightarrow \bot\})]$
where $\bot$ means no plans appropriate

```
1.animal(fox). bird(crow). object(cheese). edible(cheese).
  animal(X) :- bird(X).
2.executable(eat(A,E),[holds(A,E)])       :- animal(A),edible(E).
  executable(sing(B),[accepted(B)])       :- bird(B).
  executable(praise(fox,A), [])           :- animal(A).
  executable(grab(A,O),[holds(nobody,O)]) :- animal(A),object(O).
3.causes(sing(B),holds(nobody,O),[holds(B,O)]) :- bird(B),object(O).
  causes(eat(A,E),neg(hungry(A)),[hungry(A)])  :- animal(A),edible(E).
  causes(grab(A,O),holds(A,O),[])              :- animal(A),object(O).
4.determines(praise(fox,B),[accepted(B),declined(B)]) :- bird(B).
5.initially(holds(crow,cheese)).  initially(hungry(fox)).
6.goal([neg(hungry(fox))]).
```

Fig. 5: Fox's plans for food

i.e. first, Fox praises Crow. If Crow accepts to sing, Fox grabs the dropped cheese and eats it. Otherwise, i.e. Crow declines to sing, nothing happens. The second problem instance has the conditional plan:

$[praise(fox, crow), \textbf{cases}(\{$
$\quad \mathsf{accepted(crow)} \rightarrow [sing(crow), approach(fox, crow), attack(fox, crow)];$
$\quad \mathsf{declined(crow)} \rightarrow \bot\})]$

Thus, with the only current observation (Fox praised) Crow cannot decide which is the real intention of Fox. Since the only way to identify is an acceptance to sing, which in both cases leads to a bad consequence, losing the cheese *and/or* the territory, Crow can simply decline to sing. However, being really smart and extremely curious, she can first eat or hide the cheese in order to prevent it from falling down when singing, then she starts singing, keeping an eye on Fox's behaviors. If Fox approaches her, she flies, knowing Fox's real intention is to get her territory (supposing Crow does not get injured by a Fox attack, she can

```
1.place(tree).
2.executable(attack(fox,A),[]) :- bird(A), near(fox,A).
  executable(approach(fox,A), [happy(A)]) :- animal(A).
3.causes(attack(fox,A),occupied(fox,P),[occupied(A,P)]) :-
       animal(A),place(P).
  causes(approach(A,B),near(A,B), []) :- animal(A),animal(B).
  causes(sing(A),happy(A),[]) :- bird(A).
4.occupied(crow, tree).
5.goal([occupied(fox,tree)]).
```

Fig. 6: Fox's plan for territory

revenge on Fox to get back the territory). Otherwise, if Fox does nothing or simply goes away, Crow knows that Fox's real intention was to get the cheese.

3 Evolution Prospection

The next step, that of taking advantage of the recognized intentions gleaned from the previous stage, is implemented using Evolution Prospection Agent (EPA) system [2,3]. It enables an agent to look ahead, prospectively, into its hypothetical futures, in order to determine the best courses of evolution that satisfy its goals, and thence to prefer amongst those futures. The intentions are provided to the EPA system as goals, and EPA can help with generating the courses of evolution that achieve the goals. These courses of evolution can be provided to the intending agent as suggestions to achieve its intention (in cooperating settings) or else as a guide to prevent that agent from achieving it (in hostile settings).

We next briefly present the constructs of EPA that are necessary for the examples in this article. The whole discussion can be found in [2,3,7].

3.1 Preliminary

We next describe constructs of the evolution prospection system that are necessary for representation of the example. A full presentation can be found in [2]. The separate formalism for expressing actions can be found in [1] or [20].

3.1.1 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 is used to express that the atom is false by default (Closed World Assumption). A domain rule in $\mathcal{L}$ is a rule of the form:

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

where A is a domain atom and $L_1, \ldots, L_t$ are domain literals. An integrity constraint in $\mathcal{L}$ is a rule with an empty head. A (logic) program P over $\mathcal{L}$ is a set of domain rules and integrity constraints, standing for all their ground instances.

3.1.2 Active Goals In each cycle of its evolution the agent has a set of active goals or desires. We introduce the *on_observe*/1 predicate, which we consider as representing active goals or desires that, once triggered by the observations figuring in its rule bodies, cause the agent to attempt their satisfaction by launching all the queries standing for them, or using preferences to select them. The rule for an active goal AG is of the form:

$$on\_observe(AG) \leftarrow L_1, ..., L_t \ (t \geq 0)$$

where L_1,...,L_t are domain literals. During evolution, an active goal may be triggered by some events, previous commitments or some history-related information. When starting a cycle, the agent collects its active goals by finding all the $on\_observe(AG)$ that hold under the initial theory without performing any abduction, then finds abductive solutions for their conjunction.

3.1.3 Preferring abducibles Every program P is associated with a set of abducibles $\mathcal{A} \subseteq \mathcal{L}$. These, and their default negations, can be seen as hypotheses that provide hypothetical solutions or possible explanations to given queries. Abducibles can figure only in the body of program rules. An abducible A can be assumed 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

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

The rules about expectations are domain-specific knowledge contained in the theory of the program, and effectively constrain the hypotheses available in a situation. To express preference criteria among abducibles, we envisage an extended language $\mathcal{L}^\star$. A preference atom in $\mathcal{L}^\star$ is of the form $a \lhd b$, where a and b are abducibles. It means that if b can be assumed (i.e. considered), then $a \lhd b$ forces a to be assumed too if it can. A preference rule in $\mathcal{L}^\star$ is of the form:

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

where $L_1, ..., L_t$ are domain literals over $\mathcal{L}^\star$. This preference rule can be coded as follows:

$$expect\_not(b) \leftarrow \ L_1, ..., L_n, not\ expect\_not(a), expect(a), not\ a$$

In fact, if b is considered, the consider–rule for abducible b requires $expect\_not(b)$ to be false, i.e. every rule with the head $expect\_not(b)$ cannot have a true body. Thus, $a \lhd b$, that is if its body in the preference rule holds, and if a is expected, and not counter-expected, then a must be abduced so that this particular rule for $expect\_not(b)$ also fails, and the abduction of b may go through if all the other rules for $expect\_not(b)$ fail as well.

A priori preferences are used to produce the most interesting or relevant conjectures about possible future states. They are taken into account when generating possible scenarios (abductive solutions), which will subsequently be preferred amongst each other a posteriori.

3.1.4 A posteriori Preferences Having computed possible scenarios, represented by abductive solutions, more favorable scenarios can be preferred a posteriori. Typically, *a posteriori* preferences are performed by evaluating consequences of abducibles in abductive solutions. An *a posteriori* preference has the form:

$$A_i \ll A_j \leftarrow holds\_given(L_i, A_i),\ holds\_given(L_j, A_j)$$

where A_i, A_j are abductive solutions and L_i, L_j are domain literals. This means that A_i is preferred to A_j a posteriori if L_i and L_j are true as the side-effects of abductive solutions A_i and A_j, respectively, without any further abduction when testing for the side-effects. Optionally, in the body of the preference rule there can be any Prolog predicate used to quantitatively compare the consequences of the two abductive solutions.

3.1.5 Evolution result a posteriori preference While 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 from its present state (and any state in general). The *a posteriori* preferences are no longer appropriate, since they can be used to evaluate only one-step-far consequences of a commitment. The agent should be able to also declaratively specify preference amongst evolutions through quantitatively or qualitatively evaluating the consequences or side-effects of each evolution choice.

A posteriori preference is generalized to prefer between two evolutions. An *evolution result a posteriori* preference is performed by evaluating consequences of following some evolutions. The agent must use the imagination (look-ahead capability) and present knowledge to evaluate the consequences of evolving according to a particular course of evolution. An *evolution result a posteriori preference* rule has the form:

$$E_i \lll E_j \leftarrow holds\_in\_evol(L_i, E_i), holds\_in\_evol(L_j, E_j)$$

where E_i, E_j are possible evolutions and L_i, L_j are domain literals. This preference implies that E_i is preferred to E_j if L_i and L_j are true as evolution history side-effects when evolving according to E_i or E_j, respectively, without making further abductions when just checking for the side-effects. Optionally, in the body of the preference rule there can be recourse to any Prolog predicate, used to quantitatively compare the consequences of the two evolutions for decision making.

Example 5 (Fox-Crow, cont'd). Suppose in Fox-Crow Example 4, the final confirmed Fox's intention is that of getting food. Having recognized Fox's hidden intention, what will Crow do to prevent Fox from achieving it? The following EPA program in Figure 7 helps Crow with that.

There are two possible ways so as not to lose the Food to Fox, either simply decline to sing (but thereby missing the pleasure of singing) or hide or eat the cheese before singing.

Line 1 is the declaration of program abducibles (the last two abducibles are for use in the second phase). All of them are always expected (line 2). The counter-expectation rule in line 4 states that an animal is not expected to eat if he is full. The integrity constraints in line 5 say that Crow cannot decline to sing and sing, hide and eat the cheese, at the same time. The *a priori* preference in line 6 states that eating the cheese is always preferred to hiding it (since it may be stolen), of course, just in case eating is a possible solution (this is assured in our semantics of *a priori* preference (sub-section 3.1.3)).

```
1. abds([decline/0,sing/0,hide/2,eat/2, has_food/0,find_new_food/0]).
2. expect(decline). expect(sing). expect(hide(_,_)). expect(eat(_,_)).
3. on_observe(not_losing_cheese) <- has_intention(fox, food).
   not_losing_cheese <- decline.
   not_losing_cheese <- hide(crow,cheese), sing.
   not_losing_cheese <- eat(crow,cheese), sing.
4. expect_not(eat(A,cheese)) <- animal(A), full(A).
   animal(crow).
5. <- decline, sing.    <- hide(crow,cheese), eat(crow,cheese).
6. eat(crow,cheese) <| hide(crow,cheese).
7. no_pleasure <- decline.   has_pleasure <- sing.
8. Ai << Aj <- holds_given(has_pleasure,Ai), holds_given(no_pleasure,Aj).

9. on_observe(feed_children) <- hungry(children).
   feed_children <- has_food.   feed_children <- find_new_food.
   <- has_food, find_new_food.
10.expect(has_food) <- decline, not eat(crow,cheese).
   expect(has_food) <- hide(crow,cheese), not stolen(cheese).
   expect(find_new_food).
11.Ei <<< Ej <- hungry(children), holds_in_evol(had_food,Ei),
                holds_in_evol(find_new_food,Ej).
12.Ei <<< Ej <- holds_in_evol(has_pleasure,Ei),
                holds_in_evol(no_pleasure,Ej).

 beginProlog.
   :- assert(scheduled_events(1, [has_intention(fox,food)])),
      assert(scheduled_events(2, [hungry(children)])).
 endProlog.
```

Fig. 7: In Case Fox has intention Food

Suppose Crow is not full. Then, the counter expectation in line 4 does not hold. Thus, there are two possible abductive solutions: *[decline]* and *[eat(crow,cheese), sing]* (since the *a priori* preference prevents the choice containing *hiding*).

Next, the *a posterori* preference in line 8 is taken into account and rules out the abductive solution containing *decline* since it leads to having *no pleasure*

which is less preferred to *has pleasure* – the consequence of the second solution that contains *sing* (line 7). In short, the final solution is that Crow eats the cheese then sings, without losing the cheese to Fox and having the pleasure of singing.

Now, let us consider a smarter Crow who is capable of looking further ahead into the future in order to solve longer term goals. Suppose that Crow knows that her children will be hungry later on, in the next stage of evolution (line 9); eating the cheese right now would make her have to find new food for the hungry children. Finding new food may take long, and is always less favourable than having food ready to feed them right away (*evolution result a posteriori* preference in line 11). Crow can see three possible evolutions: $[[decline], [had\_food]]$; $[[hide(crow, cheese), sing], [had\_food]]$ and $[[eat(crow, cheese), sing], [find\_new\_food]]$. Note that in looking ahead at least two steps into the future, local preferences are not taken into account only after all evolution one were applied (full discussion can be found in [2,7]).

Now the two *evolution result a psoterirori* preferences in lines 11-12 are taken into account. The first one rules out the evolution including *finding new food* since it is less preferred than the other two which includes *had food*. The second one rules out the one including *decline*. In short, Crow will hide the food to keep for her hungry children, and still take pleasure from singing.

Note future events, like *hungry(children)*, can be asserted as Prolog code using the reserved predicate $scheduled\_events/2$. For more details of its use see [2,3].

Fig. 8: Fox and Crow Fable [1]

[1] The picture is taken from http://mythfolklore.net/aesopica/bewick/51.htm

4 Intention Recognition and Evolution Prospection for Elder Care

In the last twenty years there has been a significant increase of the average age of the population in most western countries and the number of elderly people has been and will be constantly growing. For this reason there has been a strong development of supportive technologies for elderly people living independently in their own homes, for example, RoboCare Project [10] – an ongoing project developing robots for assisting elderly people's living, SINDI – a logic-based home monitoring system [11], ILSA – an agent-based monitoring and support system for elderly [12] and PHATT – a framework developed for addressing a number of desired features for Elder Care domain [14].

For the Elder Care application domain, in order to provide contextually appropriate help for elders, it is required that the assisting system have the ability to observe the actions of the elders, recognize their intentions, and then provide suggestions on how to achieve the recognized intentions on the basis of the conceived plans. The first step of perceiving elders' actions is taken for granted. The second and third steps are addressed by our described Intention Recognition and Evolution Prospection systems, respectively. We include here an example from our previous work [4,7] for self-containment.

Example 6 (Elder Care). An elder stays alone in his apartment. The intention recognition system observes that he is looking for something in the living room. In order to assist him, the system needs to figure out what he intends to find. The possible things are: something to read (*book*); something to drink (*drink*); the TV remote control (*Rem*); and the light switch (*Switch*). The CBN representing this scenario is that of Figure 9.

4.1 Elder Intention Recognition

To begin with, we need to declare two sorts:

```
bool = {t,f}.   elder_intentions = {book,drink,rem,switch}.
```

where the second one is the sort of possible intentions of the elder. There are five top nodes, named *thirsty*(*thsty*), *like_reading*(*lr*), *like_watching*(*lw*), *tv_on*(*tv*), *light_on*(*light*), belonging to the pre-intention level to describe the causes that might give rise to the considered intentions. The values of last two nodes are observed (evidence nodes). The corresponding random attributes are declared as

```
thsty:bool. lr:bool. lw:bool. tv:bool. light:bool.
random(rth,thsty,full). random(rlr,lr,full).
random(rlw,lw,full).    random(rtv,tv,full).    random(rl, light, full).
```

and their independent probability distributions are encoded with pa-rules as

```
pa(rth,thsty(t),d_(1,2)). pa(rlr,lr(t),d_(8,10)).
pa(rlw,lw(t),d_(7,10)). pa(rtv,tv(t),d_(1,2)). pa(rl,light(t),d_(1,2)).
```

The possible intentions reading is afforded by four nodes, representing the four possible intentions of the elder, as mentioned above. The corresponding random attributes are coded specifying an attribute with domain *elder_intentions* and receives boolean values

```
i:elder_intentions --> bool.  random(ri, i(I), full).
```

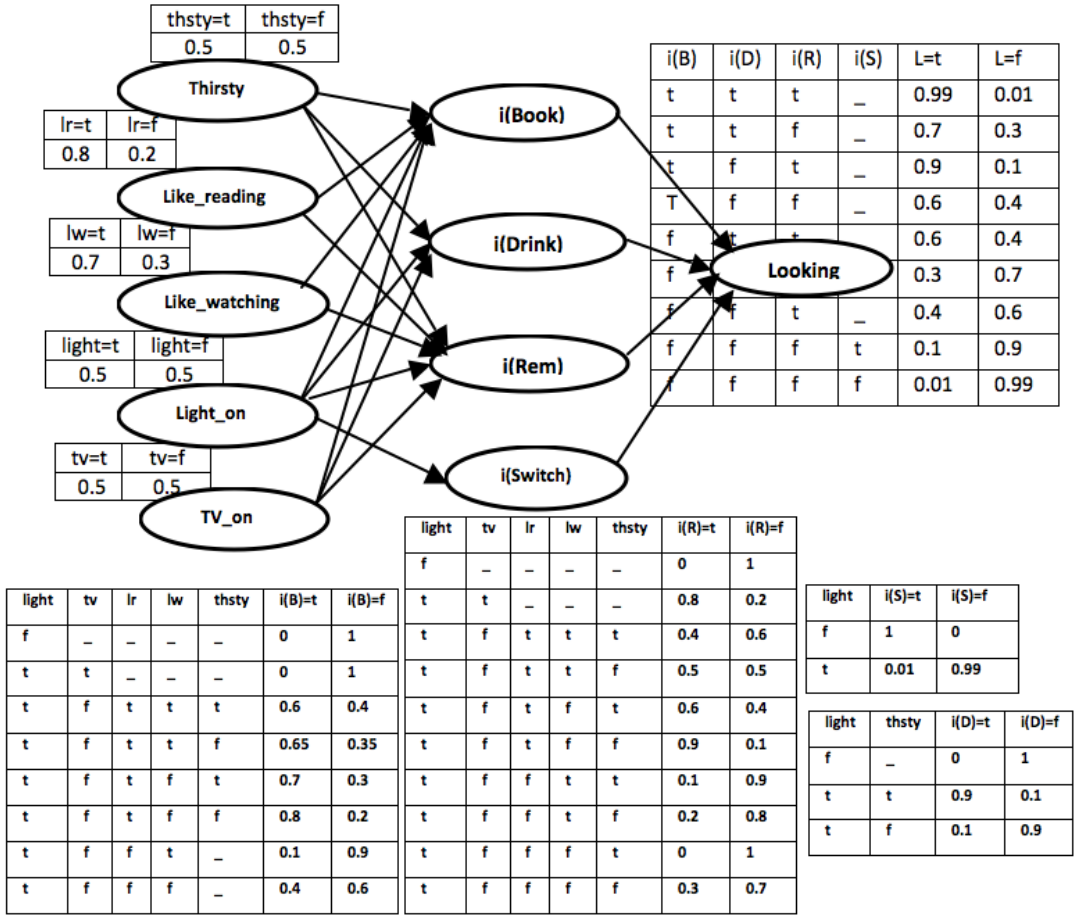

Fig. 9: Elder's intentions CBN

The probability distribution of each intention node conditional on the causes are coded in P-log below. Firstly, for $i(book)$:

```
pa(ri(book),i(book,t),d_(0,1))      :-light(f).
pa(ri(book),i(book,t),d_(0,1))      :-light(t),tv(t).
pa(ri(book),i(book,t),d_(6,10))     :-light(t),tv(f),lr(t),lw(t),thsty(t).
pa(ri(book),i(book,t),d_(65,100))   :-light(t),tv(f),lr(t),lw(t),thsty(f).
pa(ri(book),i(book,t),d_(7,10))     :-light(t),tv(f),lr(t),lw(f),thsty(t).
```

```
pa(ri(book),i(book,t),d_(8,10))    :-light(t),tv(f),lr(t),lw(f),thsty(f).
pa(ri(book),i(book,t),d_(1,10))    :-light(t),tv(f),lr(f),lw(t).
pa(ri(book),i(book,t),d_(4,10))    :-light(t),tv(f),lr(f),lw(f).
```

For $i(drink)$:

```
pa(ri(drink),i(drink,t),d_(0,1))  :- light(f).
pa(ri(drink),i(drink,t),d_(9,10)) :- light(t), thsty(t).
pa(ri(drink),i(drink,t),d_(1,10)) :- light(t), thsty(f).
```

For $i(rem)$:

```
pa(ri(rem),i(rem,t),d_(0,1))  :-light(f).
pa(ri(rem),i(rem,t),d_(8,10)) :-light(t),tv(t).
pa(ri(rem),i(rem,t),d_(4,10)) :-light(t),tv(f),lw(t),lr(t),thsty(t).
pa(ri(rem),i(rem,t),d_(5,10)) :-light(t),tv(f),lw(t),lr(t),thsty(f).
pa(ri(rem),i(rem,t),d_(6,10)) :-light(t),tv(f),lw(t),lr(f),thsty(t).
pa(ri(rem),i(rem,t),d_(9,10)) :-light(t),tv(f),lw(t),lr(f),thsty(f).
pa(ri(rem),i(rem,t),d_(1,10)) :-light(t),tv(f),lw(f),lr(t),thsty(t).
pa(ri(rem),i(rem,t),d_(2,10)) :-light(t),tv(f),lw(f),lr(t),thsty(f).
pa(ri(rem),i(rem,t),d_(0,1))  :-light(t),tv(f),lw(f),lr(f),thsty(t).
pa(ri(rem),i(rem,t),d_(3,10)) :-light(t),tv(f),lw(f),lr(f),thsty(f).
```

For $i(switch)$:

```
pa(ri(switch),i(switch,t),d_(1,1))   :- light(f).
pa(ri(switch),i(switch,t),d_(1,100)) :- light(t).
```

There is only one observation, namely, that is the elder is looking for something (*look*). The declaration of the corresponding random attribute and its probability distribution conditional on the possible intentions are given as follows:

```
look : bool.  random(rla, look, full).
pa(rla,look(t),d_(99,100)) :-i(book,t),i(drink,t),i(rem,t).
pa(rla,look(t),d_(7,10))   :-i(book,t) i(drink,t),i(rem,f).
pa(rla,look(t),d_(9,10))   :-i(book,t),i(drink,f),i(rem,t).
pa(rla,look(t),d_(6,10))   :-i(book,t),i(drink,f),i(rem,f).
pa(rla,look(t),d_(6,10))   :-i(book,f),i(drink,t),i(rem,t).
pa(rla,look(t),d_(3,10))   :-i(book,f),i(drink,t), i(rem,f).
pa(rla,look(t),d_(4,10))   :-i(book,f),i(drink,f),i(rem,t).
pa(rla,look(t),d_(1,10))   :-i(book,f),i(drink,f),i(rem,f),i(switch,t).
pa(rla,look(t),d_(1,100))  :-i(book,f),i(drink,f),i(rem,f),i(switch,f).
```

Recall that the two nodes *tv_on* and *light_on* are observed. The probabilities that the elder has the intention of looking for *book, drink, remote control* and *light switch* given the observations that he is looking around and of the states of the light (on or off) and TV (on or off) can be found in P-log with the following queries, respectively:

$?-$ pr(i(book, t) | (obs(tv(S_1)) & obs(light(S_2)) & obs(look(t))), V_1).
$?-$ pr(i(drink, t) | (obs(tv(S_1)) & obs(light(S_2)) & obs(look(t))), V_2).
$?-$ pr(i(rem, t) | (obs(tv(S_1)) & obs(light(S_2)) & obs(look(t))), V_3).

$?-$ pr(i(switch, t) | (obs(tv(S_1)) & obs(light(S_2)) & obs(look(t))), V_4).

where S_1, S_2 are boolean values (t or f) instantiated during execution, depending on the states of the light and TV. Let us consider the possible cases

- If the light is off ($S_2 = f$), then $V_1 = V_2 = V_3 = 0$, $V_4 = 1.0$, regardless of the state of the TV.
- If the light is on and TV is off ($S_1 = t, S_2 = f$), then $V_1 = 0.7521, V_2 = 0.5465, V_3 = 0.5036, V_4 = 0.0101$.
- If both light and TV are on ($S_1 = t, S_2 = t$), then $V_1 = 0, V_2 = 0.6263, V_3 = 0.9279, V_4 = 0.0102$.

Thus, if one observes that the light is off, definitely the elder is looking for the light switch, given that he is looking around. Otherwise, if one observes the light is on, in both cases where the TV is either on or off, the first three intentions *book, drink, remote control* still need to be put under consideration in the next phase, generating possible plans for each of them. The intention of looking for the light switch is very unlikely to be the case comparing with other three, thus being ruled out. When there is light one goes directly to the light switch if the intention is to turn it off, without having to look for it.

Situation-sensitive CBNs. In this scenario, the CBN may vary depending on some observed factors, for example, the time of day, the current temperature, etc. We design a logical component for the CBN to deal with those factors:

```
pa_rule(pa(rlk,lr(t),d_(0,1)),[])     :-time(T), T>0, T<5, !.
pa_rule(pa(rlk,lr(t),d_(1,10)),[])    :-time(T), T>=5, T<8, !.
pa_rule(pa(rlw,lw(t),d_(9,10)),[])    :-time(T),schedule(T,football),!.
pa_rule(pa(rlw,lw(t),d_(1,10)),[])    :-time(T), (T>23; T<5), !.
pa_rule(pa(rth,thsty(t),d_(7,10)),[]) :-temp(T), T>30, !.
pa_rule(pa(rlk,lr(t),d_(1,10)),[])    :-temp(TM), TM >30, !.
pa_rule(pa(rlw,lw(t),d_(3,10)),[])    :-temp(TM), TM>30, !.
```

When the time and temperature are defined (the assisting system should be aware of such information), they are asserted using predicates *time*/1 and *temp*/1. Note that in this modelling, to guarantee the consistency of the P-log program (there must not be two pa-rules for the same attribute instance with non-exclusive bodies) we consider time with a higher priority than temperature, enacted by using XSB Prolog *cut* operator, as can be seen in the *rlk* and *rlw* cases.

4.2 Evolution Prospection for Providing Suggestions

Having recognized the intention of another agent, EPA system can be used to provide the best courses of evolution for that agent to achieve its own intention. These courses of evolution might be provided to the other agent as suggestions.

In Elder Care domain, assisting systems should be able to provide contextually appropriate suggestions for the elders based on their recognized intentions. The assisting system is supposed to be better aware of the environment, the

elders' physical states, mental states as well as their scheduled events, so that it can provide good and safe suggestions, or simply warnings. We continue with the Elder Care example from a previous section for illustration.

Example 7 (Elder Care, cont'd). Suppose in Example 6, the final confirmed intention is that of looking for a drink. The possibilities are natural pure water, tea, coffee and juice. EPA now is used to help the elder in choosing an appropriate one. The scenario is coded with the program in Figure 10 below.

The elder's physical states are employed in *a priori* preferences and expectation rules to guarantee that only choices that are contextually safe for the elder are generated. Only after that other aspects, for example the elder's pleasure w.r.t. to each kind of drink, are taken into account, in *a posteriori* preferences.

The information regarding the environment (current time, current temperature) and the physical states of the elder is coded in the Prolog part of the program (lines 9-11). The assisting system is supposed to be aware of this information in order to provide good suggestions.

Line 1 is the declaration of program abducibles: *water, coffee, tea*, and *juice*. All of them are always expected (line 2). Line 3 picks up a recognized intention verified by the planner. The counter-expectation rules in line 4 state that *coffee* is not expected if the elder has high blood pressure, experiences difficulty to sleep or it is late; and juice is not expected if it is late. Note that the reserved predicate *prolog/1* is used to allow embedding prolog code in an EPA program. More details can be found in [2,3]. The integrity constraints in line 5 say that is is not allowed to have at the same time the following pairs of drink: tea and coffee, tea and juice, coffee and juice, and tea and water. However, it is the case that the elder can have coffee or juice together with water at the same time.

The *a priori* preferences in line 6 say in the morning coffee is preferred to tea, water and juice. And if it is hot, juice is preferred to all other kinds of drink and water is preferred to tea and coffee (line 7). In addition, the *a priori* preferences in line 8 state if the weather is cold, tea is the most favorable, i.e. preferred to all other kinds of drink.

Now let us look at the suggestions provided by the Elder Care assisting system modelled by this EPA program, considering some cases:

1. time(24) (*late*); temperature(16) (*not hot, not cold*); no high blood pressure; no sleep difficulty: there are two a priori abductive solutions: [*tea*], [*water*]. Final solutions: [*tea*] (since it has greater level of pleasure than water, which is ruled out by the *a posteriori* preference in line 12).
2. time(8) (*morning time*); temperature(16) (*not hot, not cold*); no high blood pressure; no sleep difficulty: there are two abductive solutions: [*coffee*], [*coffee*, *water*]. Final: [*coffee*], [*coffee*, *water*].
3. time(18) (*not late, not morning time*); temperature(16) (*not cold, not hot*); no high blood pressure; no sleep difficulty: there are six abductive solutions: *[coffee]*, *[coffee,water]*, *[juice]*, *[juice,water]*, *[tea]*, and *[water]*. Final: *[coffee]*, *[coffee,water]*.

```
1. abds([water/0, coffee/0, tea/0, juice/0]).
2. expect(coffee). expect(tea). expect(water). expect(juice).
3. on_observe(drink)   <- has_intention(elder,drink).
   drink <- tea. drink <- coffee. drink <- water. drink <- juice.
4. expect_not(coffee) <-  prolog(blood_high_pressure).
   expect_not(coffee) <- prolog(sleep_difficulty).
   expect_not(coffee) <- prolog(late).
   expect_not(juice)  <- prolog(late).
5. <- tea, coffee.    <- coffee, juice.
   <- tea, juice.     <- tea, water.
6. coffee <| tea    <- prolog(morning_time).
   coffee <| water  <- prolog(morning_time).
   coffee <| juice  <- prolog(morning_time).
7. juice <| coffee <- prolog(hot).  juice <| tea    <- prolog(hot).
   juice <| water  <- prolog(hot).  water <| coffee <- prolog(hot).
   water <| tea    <- prolog(hot).
8. tea <| coffee <- prolog(cold).  tea <| juice <- prolog(cold).
   tea <| water  <- prolog(cold).
9. pleasure_level(3) <- coffee. pleasure_level(2) <- tea.
   pleasure_level(1) <- juice.  pleasure_level(0) <- water.
10.sugar_level(1) <- coffee. sugar_level(1) <- tea.
   sugar_level(5) <- juice.  sugar_level(0) <- water.
11.caffein_level(5) <- coffee.  caffein_level(0) <- tea.
   caffein_level(0) <- juice.   caffein_level(0) <- water.
12.Ai << Aj <- holds_given(pleasure_level(V1), Ai),
               holds_given(pleasure_level(V2), Aj), V1 > V2.

13.on_observe(health_check) <- time_for_health_check.
   health_check <- precise_result.
   health_check <- imprecise_result.
14.expect(precise_result) <- no_hight_sugar, no_high_caffein.
   expect(imprecise_result).
   no_high_sugar   <- sugar_level(L), prolog(L < 2).
   no_high_caffein <- caffein_level(L), prolog(L < 2).
15.Ei <<< Ej <- holds_in_evol(precise_result, Ei),
                holds_in_evol(imprecise_result, Ej).

 beginProlog.
   :- assert(scheduled_events(1, [has_intention(elder,drink)])),
      assert(scheduled_events(2, [time_for_health_check])).
   late         :- time(T), (T > 23; T < 5).
   morning_time :- time(T), T > 7, T < 10.
   hot  :- temperature(TM), TM > 32.
   cold :- temperature(TM), TM < 10.
   blood_high_pressure :- physical_state(blood_high_pressure).
   sleep_difficulty    :- physical_state(sleep_difficulty).
 endProlog.
```

Fig. 10: Elder Care: Suggestion for a Drink

4. time(18) *(not late, not morning time)*; temperature(16) (*not cold, not hot*); high blood pressure; no sleep difficulty: there are four abductive solutions: *[juice], [juice,water], [tea]*, and *[water]*. Final: [*tea*].
5. time(18) (*not late, not morning time*); temperature(16) (*not cold, not hot*); no high blood pressure; sleep difficulty: there are four abductive solutions: *[juice], [juice,water], [tea]*, and *[water]*. Final: [*tea*].
6. time(18) (*not late, not morning time*); temperature(8) (*cold*); no high blood pressure; no sleep difficulty: there is only one abductive solution: *[tea]*.
7. time(18) (*not late, not morning time*); temperature(35) (*hot*); no high blood pressure; no sleep difficulty: there are two abductive solutions: *[juice], [juice,water]*. Final: *[juice], [juice,water]*.

If the *evolution result a posteriori preference* in line 15 is taken into account and the elder is scheduled to go to the hospital for health check in the second day: the first and the second cases do not change. In the third case: the suggestions are [*tea*] and [*water*] since the ones that have *coffee* or juice would cause high caffein and sugar levels, respectively, which can make the checking result (health) imprecise (lines 13-15). Similarly for all the other cases ...

Note future events can be asserted as Prolog code using the reserved predicate *schedule_events*/2. For more details of its use see [2,3].

As one can gather, the suggestions provided by this assisting system are quite contextually appropriate. We might elaborate current factors (time, temperature, physical states) and even consider more factors to provide more appropriate suggestions if the situation gets more complicated.

5 Conclusions and Future Work

We have shown a coherent LP-based system addressing the overall process from recognizing intentions of an agent to taking advantage of those intentions in dealing with the agent, either in a cooperating or hostile settings. The intention recognition part is achieved by means of an articulate combination of situation-sensitive CBNs and a plan generator. Based on the situation at hand and a starting CBN default for the problem domain, its situation-sensitive version is dynamically reconfigured, using LP techniques, in order to compute the likelihood of intentions w.r.t. the situation given, then filter out those much less likely than others. The computed likelihoods enable the recognizing agent to focus on the more likely ones, which is especially important for when having to make a quick decision. Henceforth, the plan generator just needs to work on the remaining relevant intentions. In addition, we have shown how generated plans can guide the recognition process: which actions (or their effects) should be checked for whether they were (hiddenly) executed by the intending agent. We have illustrated all these features with the Fox and Crow example.

We have also shown a LP-based system for assisting elderly people based on the described intention recognizer and Evolution Prospection system. The recognizer is to figure out intentions of the elders based on their observed actions or the effects their actions have in the environment. The implemented Evolution

Prospection system, being aware of the external environment, elders' preferences and their note future events, is then employed to provide contextually appropriate suggestions that achieve the recognized intention. The system built-in expectation rules and a priori preferences take into account the physical state (health reports) information of the elder to guarantee that only contextually safe healthy choices are generated; then, information such as the elders pleasure, interests, scheduled events, etc. are taken into account by *a posteriori* and *evolution result a posteriori preferences*.

We believe to have exhibited the usefulness and advantage of our approach of combining several needed features to tackle the Elder Care application domain, by virtue of an integrated logic programming approach. One future direction is to implement meta-explanation about evolution prospection [26]. It would be quite useful in the considered setting, as the elder care assisting system should be able to explain to elders the whys and wherefores of suggestions made. Moreover, it should be able to produce the abductive solutions found for possible evolutions, keeping them labeled by the preferences used (in a partial order) instead of exhibiting only the most favorable ones. This would allow for final preference change on the part of the elder.

There are currently several other possible future directions to explore. First of all, we can employ an interplay between CBNs and the planner. Besides being a consumer of CBNs as shown, the planner can also be a producer for the CBN in the following ways. Firstly, its feedback about the plausible final intention of the intending agent may increase the corresponding probabilistic relations of the confirmed intention in the CBN; secondly, when new actions (or their effects) of the intending agent, not observed before, become confirmed, the CBN is updated again, which might rule out more intentions, not yet explored nor able to be confirmed or denied. Moreover, the planner might do real experiments, or even thought experiments, where values of nodes may be enforced true. The thought experiments may involve hypothetical or even counterfactual reasoning (possibly prospecting the future [2]).

In addition, the advance in LP semantics for evolving program with updates [27] should be used to give more flexibility in updating CBNs with new information. This is essential when more dynamic reasoning processes, e.g. in the above CBNs-Planner interplay, are employed. To this end, we also plan to parameterize P-log, i.e. enable P-log to have variables in different constructs, such as sort declarations, probabilistic information pa-rules, etc., and those variables be provided by the program calling it, depending on the context. CBNs updating would very much benefit from this ability.

Clearly, an agent recognizes the intention of another for a purpose, i.e. Intention Recognition should be purposive. The depth of understanding of an intention that is required by an agent depends on why knowledge of the intention is required. It might be that an agent needs only the broadest understanding of the intention. In that case, simply knowing the general class of intention is adequate and details are unimportant. But it also might be that details are important for a given purpose. For example, in the Elder Care example (Example

6), confirming that the elder has an intention of looking for something is not enough; the details about what he/she is looking for are necessary for the purpose of providing appropriate support or suggestions. To this end, we plan to use an ontology of intentions, and the more general intentions are discovered earlier. Confirmation of a general intention may trigger the discovery of more specific ones, if more details of understanding of the intention are required and available. Actually, richer details of understanding an intention might come up during the recognition process of a general intention as more observations can be gathered. An example of this arises in the Fox-Crow example (Example 1). Initially, the system is trying to recognize a general intention: if Fox intends to get some food; and during the recognition process, the detail that Fox's intention is to get a concrete kind of food, Crow's cheese, is found out. However, this is not always the case. For instance, in the Elder Care example, confirming that the elder is looking for something simply triggers a new Intention Recognition process, including the design of a new CBN for computing the likelihood of specific intentions (a drink, a book, TV remote control or light switch) and generating plans for the likely ones, so as to figure out more details about the intention. We plan to attempt to categorize the possible cases and conduct appropriate techniques for each of them.

References

1. L. M. Pereira, H. T. Anh. *Intention Recognition via Causal Bayes Networks plus Plan Generation*, in: Seabra Lopes, L.; Lau, N.; Mariano, P.; Rocha, L.M. (eds.), Progress in Artificial Intelligence, Procs. 14th Portuguese Intl.Conf. on Artificial Intelligence (EPIA'09), pp. 138-149, Springer LNAI 5816, October 2009.
2. L. M. Pereira, H. T. Anh. *Evolution Prospection*, in: K. Nakamatsu (ed.), Procs. Intl. Symposium on Intelligent Decision Technologies (KES-IDT'09), pages 51-63, Springer Studies in Computational Intelligence 199, 2009.
3. L. M. Pereira, H. T. Anh. *Evolution Prospection in Decision Making*, in: Intelligent Decision Technologies (IDT), 3(3):157–171, 2009.
4. L. M. Pereira, H. T. Anh. *Elder Care via Intention Recognition and Evolution Prospection.* in: S. Abreu, D. Seipel (eds.), Procs. 18th Intl. Conf. on Applications of Declarative Programming and Knowledge Management (INAP'09), 2009.
5. C. Baral, M. Gelfond, and N. Rushton. *Probabilistic reasoning with answer sets.* In Procs. Logic Programming and Nonmonotonic Reasoning (LPNMR 7), pages 21–33, Springer LNAI 2923, 2004.
6. H. T. Anh, C. K. Ramli, C. V. Damásio. *An implementation of extended P-log using XASP*, in: M. Garcia de la Banda, E. Pontelli (eds.), In Procs. Intl. Conf. Logic Programming, pp. 739–743, Springer LNCS 5366, 2008.
7. H. T. Anh. *Evolution Prospection with Intention Recognition via Computational Logic.* Master Thesis, Technical University of Dresden, June 2009.
8. C. Baral, M. Gelfond, N. Rushton. *Probabilistic reasoning with answer sets.* Theory and Practice of Logic Programming, 9(1): 57-144, January 2009.
9. 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
10. A. Cesta, F. Pecora. *The Robocare Project: Intelligent Systems for Elder Care.* AAAI Fall Symposium on Caring Machines: AI in Elder Care, USA 2005.

11. A. Mileo, D. Merico, R. Bisiani. *A Logic Programming Approach to Home Monitoring for Risk Prevention in Assisted Living* , ICLP, Springer LNCS 5366, 2008.
12. K. Z. Haigh , L. M. Kiff , J. Myers , V. Guralnik , C. W. Geib , J. Phelps , T. Wagner . *The independent lifestyle assistant (i.l.s.a.): Ai lessons learned.* In Procs. of Conf. on Innovative Applications of AI, 852857, 2004.
13. M. V. Giuliani, M. Scopelliti, F. Fornara. *Elderly people at home: technological help in everyday activities.* IEEE International Workshop on In Robot and Human Interactive Communication, pp. 365-370, 2005.
14. C. W. Geib. *Problems with intent recognition for elder care.* In Procs. AAAI Workshop Automation as Caregiver, 2002.
15. C. Heinze. *Modeling Intention Recognition for Intelligent Agent Systems*, Doctoral Dissertation, the University of Melbourne, Australia, 2003.
16. K. A. Tahboub. *Intelligent Human-Machine Interaction Based on Dynamic Bayesian Networks Probabilistic Intention Recognition.* J. Intelligent Robotics Systems, vol. 45, no. 1, pages 31-52, 2006.
17. O. C. Schrempf, D. Albrecht, U. D. Hanebeck. *Tractable Probabilistic Models for Intention Recognition Based on Expert Knowledge*, In Procs. 2007 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS 2007), pages 1429–1434, 2007.
18. H. A. Kautz, J. F. Allen. *Generalized plan recognition.* In Procs. 1986 Conf. of the American Association for Artificial Intelligence, AAAI 1986: 32-37, 1986.
19. T. Eiter, W. Faber, N. Leone, G. Pfeifer, A. Polleres. *A Logic Programming Approach to Knowledge State Planning, II: The DLV^{K} System.* Artificial Intelligence 144(1-2): 157-211, 2003.
20. P. H. Tu, T. C. Son, C. Baral. *Reasoning and Planning with Sensing Actions, Incomplete Information, and Static Causal Laws using Answer Set Programming.* Theory and Practice of Logic Programming, 7(4): 377-450, July 2007.
21. An implementation of ASCP using XASP available at: http://centria.di.fct.unl.pt/ *lmp*/software/cataplan-online.zip
22. M. Gelfond, V. Lifschitz, *Representing actions and change by logic programs.* Journal of Logic Programming 17, 2,3,4, 301–323, 1993.
23. C. Glymour. *The Mind's Arrows: Bayes Nets and Graphical Causal Models in Psychology.* MIT Press, 2001.
24. B. Kowalski. *How to be Artificially Intelligent*, online book. Downloadable at: http://www.doc.ic.ac.uk/ rak/
25. J. Pearl. *Causality: Models, Reasoning, and Inference.* Cambridge U.P., 2000.
26. L. M. Pereira, A. M. Pinto. *Inspection Points and Meta-Abduction in Logic Programs*, in: S. Abreu, D. Seipel (eds.), Procs. 18th Intl. Conf. on Applications of Decl. Programming and Knowledge Management (INAP'09), pp. 171–184, 2009.
27. J. J. Alferes, A. Brogi, J. A. Leite, L.M. Pereira. *Evolving logic programs.* Procs. 8th Europ. Conf. on Logics in AI (JELIA'02), pp. 50–61, Springer LNAI 2424, 2002.
28. J. J. Alferes, F. Banti, A. Brogi, J. A. Leite. *The Refined Extension Principle for Semantics of Dynamic Logic Programming* , Studia Logica 79(1): 7-32, 2005.
29. J. J. Alferes, J. A. Leite, L. M. Pereira, H. Przymusinska, T. C. Przymusinski. *Dynamic updates of non-monotonic knowledge bases.* J. Logic Programming, 45(1-3):4370, 2000.
30. I. Niemelä, P. Simons. *Smodels: An implementation of the stable model and well-founded semantics for normal logic programs.* 4th Intl. Conf. on Logic Programming and Nonmonotonic Reasoning, Springer LNAI 1265, pages 420–429, 1997.
31. T. Swift. *Tabling for non-monotonic programming.* Annals of Mathematics and Artificial Intelligence, 25(3–4):201-240, 1999.
32. *The XSB System Version 3.0 Vol. 2: Libraries, Interfaces and Packages.* July 2006.