Elder Care via Intention Recognition and Evolution Prospection

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

Centro de Inteligência Artificial (CENTRIA)
Universidade Nova de Lisboa, 2829-516 Caparica, Portugal

Abstract. We explore and exemplify the application in the Elder Care context of the ability to perform Intention Recognition and of wielding Evolution Prospection methods. This is achieved by means of an articulate use of Causal Bayes Nets (for heuristically gauging probable general intentions), combined with specific generation of plans involving preferences (for checking which such intentions are plausibly being carried out in the specific situation at hand). 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 Nets, P-Log, Evolution Prospection, Preferences.

1 Introduction

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 technology for elderly people living independently in their own homes, for example, RoboCare Project [8] – an ongoing project developing robots for assisted elderly people's living, SINDI – a logic-based home monitoring system [9].

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. In this paper we focus on the latter two steps in order to design and implement an elder care logic programming based assisting system. The first step of perceiving elders' actions is taken for granted. For elders' intention recognition based on their observable actions, we employ our work on Intention Recognition system using Causal Bayes Networks and plan generation techniques, described in [1]. The intention recognition component is indispensable for living-alone elders, in order to provide them with timely suggestions. The next step, that of providing action suggestions for realizing the recognized intention gleaned from the previous stage, is implemented using our Evolution Prospection Agent (EPA) system [2,3]. The latter can prospectively look ahead into the future to choose the best course of evolution whose actions achieve the recognized intention, while being aware

of the external environment and of an elder's preferences and already scheduled future events. 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 elder's pleasure, interests, etc. are taken into account by a posteriori preferences and the like. The advance and easiness of expressing preferences in EPA [3] enable to closely take into account the elders' preferences, which we believe, would increase the degree of acceptance of the elders w.r.t. the technological help - an important issue of the domain [10].

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 [11], 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 [11,14].

One of the main 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 paper, we use an approach to solve this problem employing so-called *situation-sensitive Causal Bayes Networks* (CBN) - That is, CBNs [18] 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 intentions. 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 [4,6,5], 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 those using solely BNs [12,13] 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. The approach of combining 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. 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 [16] from XSB Prolog using the XASP package [7,24] for interfacing with Smodels [22] – an answer set solver – or, alternatively, rely on plan libraries so obtained.

In the sequel we briefly describe the intention recognition and evolution prospection systems, but not the planner. Then we show in detail how to combine them to provide contextual help for elderly people, illustrating with an extended example.

2 Intention Recognition

2.1 Causal Bayes Networks

A Bayes Network (BN), recapitulated here for convenience in order to help see their realization in P-log, is a pair consisting of a directed acyclic graph (dag) whose nodes represent variables and missing edges encode conditional independencies between the variables, and an associated probability distribution satisfying the assumption of conditional independence (Causal Markov Assumption - CMA), saying that variables are independent of their non-effects conditional on their direct causes [18].

If there is an edge from node A to another node 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 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*. 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, one without parents, the unconditional distribution of the variable is specified. These distributions are called Conditional Probability Distribution (CPD) of the BN.

Suppose the nodes of the dag form a causally sufficient set [17], i.e. no common causes of any two nodes are omitted, then implied by CMA [17], the joint distribution of all node values of the set 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

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

where the numerator and denominator are computed by summing the joint probabilities over all absent variables with respect to 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 specify the structure of the network, its Conditional Probability Distribution (CPD) and the prior probability distribution of the top nodes.

2.2 Intention recognition with Causal Bayesian Networks

The first phase of the intention recognition system is to find out how likely each possible 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. It is carried out by using a CBN with nodes standing for binary random variables that represent causes, intentions, actions and effects.

Intentions are represented by intermediate nodes whose ancestor nodes stand for causes that give rise to intentions. Intuitively, we extend Heinze's tri-level model [11] with a so-called pre-intentional level that describes the causes of intentions, used to estimate prior probabilities of the intentions. This additional level guarantees the causal sufficiency condition of the set of nodes of the dag. However, if these prior probabilities can be specified without considering the causes, intentions are represented by top nodes. These reflect the problem context or the intending agent's mental state.

Observed 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 causal relations among nodes of the BNs (e.g. which causes give rise to an intention, which intentions trigger an action, which actions have an effect), as well as their CPD and the distribution of the top nodes, are specified by domain experts. However, they might be learnt mechanically. 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 (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 1. Its CPD and the distribution of top nodes will be given directly in P-log code.

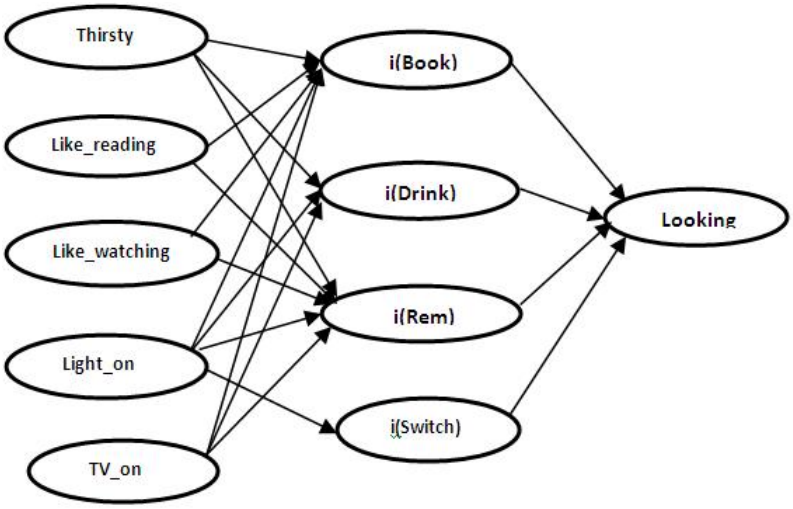

Fig. 1: Elder's intentions CBN

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 [4,6] 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 [7] makes the P-log system sometimes computationally redundant. Newer developments of P-log [5] use the XASP package of XSB Prolog [24] for interfacing with Smodels [22] – 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 [5]. Moreover, the tabling mechanism of XSB [23] 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 paatoms) 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 An example: recognizing elders' intentions

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).
```

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:

$$? - \ pr(i(book, t)\ '|'\ (obs(tv(S_1))\ \&\ light(S_2)\ \&\ obs(look(t))), V_1).$$
$$? - \ pr(i(drink, t)\ '|'\ (obs(tv(S_1))\ \&\ light(S_2)\ \&\ obs(look(t))), V_2).$$
$$? - \ pr(i(rem, t)\ '|'\ (obs(tv(S_1))\ \&\ light(S_2)\ \&\ obs(look(t))), V_3).$$
$$? - \ pr(i(switch, t)\ '|'\ (obs(tv(S_1))\ \&\ light(S_2)\ \&\ obs(look(t))), V_4).$$

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.

2.4.1 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. 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 introduced into a CBN which is 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. including integrity constraint satisfaction, contradiction removal, preferences, or inductive learning, whose results can be compiled (in part) into an evolving CBN.

The general issue of how to update a CBN with new probabilistic information can take advantage of the advances in LP semantics for evolving programs by means of rule updates [19,20,21]. In this paper, however, we don't need such general updates, and make do with a simpler way, shown in the sequel.

Example 2 (Elder Care, cont'd). 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, !.
```

Given P-log probabilistic information by $pa/3$ rules, then the corresponding so-called situation-sensitive $pa\_rule/2$ predicate takes the head and body of some $pa/3$ rule as its first and second arguments, respectively, and includes conditions for its activation in its own body. Now, a situation is given by asserted facts representing it and, in order to find the probabilistic information specific to the given situation, we simply use the XSB Prolog built-in $findall/3$ predicate to find all true $pa/3$ literals expressed by the $pa\_rule/2$ rules with true bodies in the situation. For example, 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.

2.5 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. The plan generation phase can be carried out by ASCP – an ASP logic programming based conditional planner [16], as in our work [1], which details the method and the language for expressing actions. For lack of space, we will simply assume here that the system has been

furnished a plan library that provides recipes for achieving the recognized intentions in the situation. This alternative method has also been employed in several works of plan recognition, e.g. [11,14].

3 Elder Assisting with Evolution Prospection

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 [16].

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.

3.2 Evolution Prospection as An Intention Consumer

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 3 (Elder Care, cont'd). Suppose in Example 1, 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 2 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.

```
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. 2: Elder Care: Suggestion for a Drink

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]*.
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 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.

4 Conclusions and Future Work

We have shown a coherent LP-based system for assisting elderly people based on an 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, via a combination of situation-sensitive Causal Bayes Nets and a planner. 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 elder's pleasure, interests, scheduled events, etc. are taken into account by *a posteriori* and *evolution result a posteriori* preferences.

We believe to have shown the usefulness and advantage of our approach of combining several needed features to tackle the application domain, by virtue of an integrated logic programming approach.

One future direction is to implement meta-explanation about evolution prospection. 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.

References

1. L. M. Pereira, H. T. Anh. *Intention Recognition via Causal Bayes Networks plus Plan Generation*, Procs. 14th Portuguese Conf. on AI (EPIA'09), Springer LNAI, October 2009 (to appear).
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*. Intl. Journal of Intelligent Decision Technologies, IOS Press (to appear in 2009).
4. 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.
5. H. T. Anh, C. K. Ramli, C. V. Damásio. *An implementation of extended P-log using XASP.* In Procs. Intl. Conf. Logic Programming, Springer LNCS 5366, 2008.
6. C. Baral, M. Gelfond, N. Rushton. *Probabilistic reasoning with answer sets.* Theory and Practice of Logic Programming, 9(1): 57-144, January 2009.
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. A. Cesta, F. Pecora. *The Robocare Project: Intelligent Systems for Elder Care*. AAAI Fall Symposium on Caring Machines: AI in Elder Care, USA 2005.
9. A. Mileo, D. Merico, R. Bisiani. *A Logic Programming Approach to Home Monitoring for Risk Prevention in Assisted Living* . In Procs. Intl. Conf. Logic Programming, Springer LNCS 5366, 2008.
10. 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.

11. C. Heinze. *Modeling Intention Recognition for Intelligent Agent Systems*, Doctoral Thesis, the University of Melbourne, Australia, 2003. Online available: $http : //www.dsto.defence.gov.au/publications/scientific\_record.php?record = 3367$
12. 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.
13. 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.
14. H. A. Kautz and J. F. Allen. *Generalized plan recognition.* In Procs. 1986 Conf. of the American Association for Artificial Intelligence, AAAI 1986: 32-37, 1986.
15. T. Eiter, W. Faber, N. Leone, G. Pfeifer, A. Polleres. *A Logic Programming Approach to Knowledge State Planning, II: The $DLV^{\mathcal{K}}$ System*. Artificial Intelligence 144(1-2): 157-211, 2003.
16. 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.
17. C. Glymour. *The Mind's Arrows: Bayes Nets and Graphical Causal Models in Psychology*. MIT Press, 2001.
18. J. Pearl. *Causality: Models, Reasoning, and Inference.* Cambridge U.P., 2000.
19. J. J. Alferes, A. Brogi, J. A. Leite, L.M. Pereira. *Evolving logic programs.* Procs. 8th European Conf. on Logics in AI (JELIA'02), pages 50−61, Springer LNAI 2424, 2002.
20. 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.
21. 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.
22. 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.
23. T. Swift. *Tabling for non-monotonic programming.* Annals of Mathematics and Artificial Intelligence, 25(3–4):201-240, 1999.
24. *The XSB System Version 3.0 Vol. 2: Libraries, Interfaces and Packages.* July 2006.