Intention Recognition via Causal Bayes Networks plus Plan Generation

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. In this paper, 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 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.

Keywords: Intention recognition, Causal Bayes Networks, Plan generation, P-log, ASCP, Logic Programming.

1 Introduction

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 [5], 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 in the environment. According to this definition, an 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 [5,8].

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 come up with 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 propose an approach to solve this problem using so-called *situation-sensitive Causal Bayes Networks* (CBN) - That is, CBNs [15] that change according to the situation under consideration, itself subject to change. Therefore, in some given situation, a CBN is configured, dynamically, to compute the likelihood of

intentions and filter out the much less likely intentions. The plan generator then only needs to deal with the remaining (relevant) intentions. Moreover, it being one of the important advantages of our approach, on the basis of the information provided by the CBN the recognizing agent can see which intentions are more likely and worth addressing first, and thus, in case of having to make a quick decision, focus on the most relevant ones.

CBNs, in our work, are represented in P-log [1,3,2], 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 updated 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 [6,7] 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 for whether they were (hiddenly) executed by the intending agent. We can make use of any plan generators available. In this work, for integration's sake, we use the ASP based conditional planner called ASCP [10], re-implemented [11] in XSB Prolog using the XASP package [4,22] for interfacing with Smodels [20] – an answer set solver.

The rest of the paper is organized as follows. Section 2 briefly recalls CBNs and describes how they are used for intention recognition. This section also briefly introduces P-log. Section 3 proceeds by illustrating P-log with an example and discusses situation-sensitive CBNs. Section 4 describes the ASCP planner and shows how it is used for generating plans achieving hypothesized intentions. The paper ends with conclusions and directions for the future.

2 Causal Bayes Networks in P-log

2.1 Causal BN

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

A Bayes Network 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 [15].

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, 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 [14], i.e. no common causes of any two nodes are omitted, then implied by CMA [14], 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

$$P(X|O_1, O_2, ..., O_m) = \frac{P(X, O)}{P(O)} = \frac{P(X, O_1, ..., O_m)}{P(O_1, ..., O_m)} \tag{1}$$

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, finally, 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 observed or unobserved, 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, and adopts the following structure.

Intentions are represented by intermediate nodes whose ancestor nodes represent causes that give rise to those intentions. Intuitively, we extend Heinze's tri-level model [5] with a so-called pre-intentional level that describes the causes of intentions, which are used to estimate prior probabilities of the intentions. This additional level also 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. Top nodes 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 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 (The Fox-Crow story - adapted from Aesop's fable). 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 1. 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 ($Praised$).

2.3 P-log

The computation in CBNs can be automated by using P-log, a declarative language that combines logical and probabilistic reasoning, and uses ASP as its logical and CBNs as its probabilistic foundations.

The original P-log [1,3] 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 [4] makes the P-log system sometimes computationally redundant. Newer developments of P-log [2] use the XASP package of XSB Prolog [22] for interfacing with Smodels [20] – 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

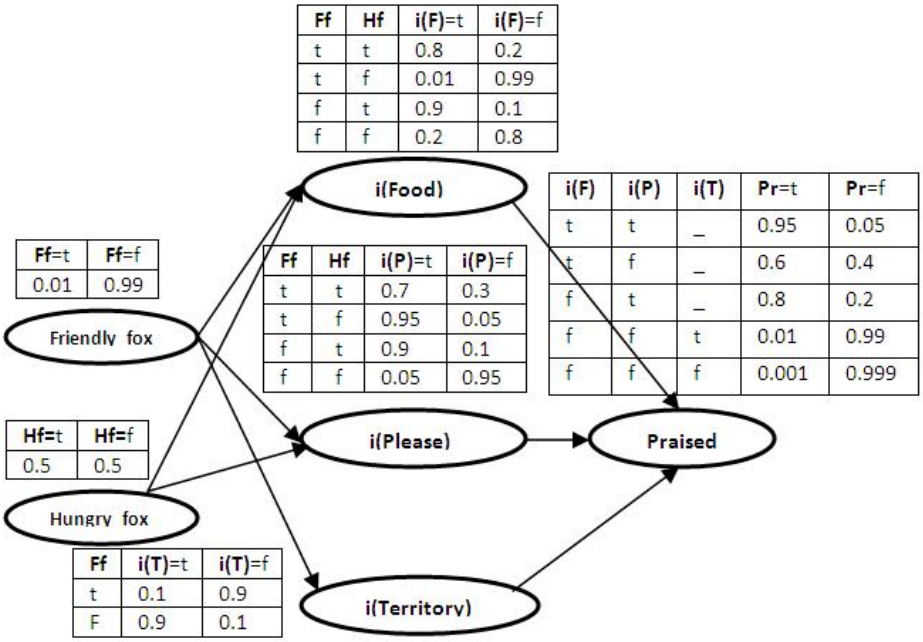

Fig. 1: Fox's intentions CBN

(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 [2]. Moreover, the tabling mechanism of XSB [21] significantly improves the performance of the system.

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

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 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 $c = \{x_1, ..., x_n\}$, by 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:

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

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

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

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 $DynamicRange$ 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 $range(a)$.

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 $\frac{A}{B}$.

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. 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.

3 Recognizing Fox's intentions - An Example

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

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 1 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 hungry are both defined by default, thus, equal to $1/2$.

```
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. 2: Fox's intentions CBN

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

$?-\ \mathbf{pr}(\mathbf{i}(\mathbf{food},\mathbf{t})\ '|'\ \mathbf{obs}(\mathbf{praised}(\mathbf{t})),\mathbf{V_1})$. The answer is: $V_1 = 0.9317$.

$?-\ \mathbf{pr}(\mathbf{i}(\mathbf{territory},\mathbf{t})\ '|'\mathbf{obs}(\mathbf{praised}(\mathbf{t})),\mathbf{V_2})$. The answer is: $V_2 = 0.8836$.

$?-\ \mathbf{pr}(\mathbf{i}(\mathbf{please},\mathbf{t})\ '|'\ \mathbf{obs}(\mathbf{praised}(\mathbf{t})),\mathbf{V_3})$. The answer is: $V_3 = 0.0900$.

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.

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. including 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 [17,18,19]. 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 2. 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 [2]).

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 $food$.

4 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 [9] and ASCP – an ASP based conditional planner [10].

In our system, plan generation is carried out by a new implementation of ASCP in XSB Prolog using XASP package [11]. 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 [10].

4.1 Action language A_K^c

ASCP uses A_K^c - a representation action language that extends $\mathcal{A}$ [12] 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 **A** and a set of fluents **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 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 [10] for details).

4.2 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 [13].

Example 4 (Fox-Crow (cont'd)). The scenarios with intentions of getting food and territory are represented in Figure 3 and 4, respectively. The first problem instance has the conditional plan:

$[praise(fox, crow), \mathbf{cases}(\{$
$\quad \mathsf{accepted}(\mathsf{crow}) \rightarrow [sing(crow), grab(fox, cheese), eat(fox, cheese)];$
$\quad \mathsf{declined}(\mathsf{crow}) \rightarrow \bot\})]$ (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. 3: 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:

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

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

```
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. 4: Fox's plan for territory

flies, knowing Fox's real intention is to get her territory (supposing Crow does not get injured by a Fox attack, she can 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.

5 Conclusions and Future Work

We have shown a novel approach to intention recognition, by combining 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 re-configured, 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 an example.

There are currently several 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 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 being observable before, become confirmed, the CBN is updated again, which might rule out more intentions, not yet explored or 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 [16]).

In addition, the advance in LP semantics for evolving program with updates [17] 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.

References

1. C. Baral, M. Gelfond, and N. Rushton. *Probabilistic reasoning with answer sets.* In Procs. LPNMR7, pages 21–33, LNAI 2923, 2004.
2. H. T. Anh, C. K. Ramli, C. V. Damásio. *An implementation of extended P-log using XASP.* In Procs. Intl. Conf. Logic Programming, LNCS 5366, Udine, Italy, 2008.
3. C. Baral, M. Gelfond, N. Rushton. *Probabilistic reasoning with answer sets.* TPLP, Volume 9, Part 1, pages 57–144, January 2009.
4. 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
5. C. Heinze. *Modeling Intention Recognition for Intelligent Agent Systems*, Doctoral Thesis, the University of Melbourne, Australia, 2003. Online available: $http : //www.dsto.defence.gov.au/publications/scientific\_record.php?record = 3367$
6. K. A. Tahboub, *Intelligent Human-Machine Interaction Based on Dynamic Bayesian Networks Probabilistic Intention Recognition.* Journal of Intelligent Robotics Systems, vol. 45, no. 1, pages 31-52, 2006.
7. 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.
8. H. A. Kautz and J. F. Allen. *Generalized plan recognition.* In Procs. 1986 Conf. of the American Association for Artificial Intelligence, 1986.
9. 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, 157-211, 2003.
10. P. H. Tu, T. C. Son, C. Baral. *Reasoning and Planning with Sensing Actions, Incomplete Information, and Static Causal Laws using Answer Set Programming.* TPLP, Volume 7, Issue 4, July 2007.
11. An implementation of ASCP using XASP available at: http://centria.di.fct.unl.pt/ lmp/software/cataplan-online.zip
12. M. Gelfond, V. Lifschitz, *Representing actions and change by logic programs.* Journal of Logic Programming 17, 2,3,4, 301–323, 1993.
13. B. Kowalski. *How to be Artificially Intelligent*, online book. Downloadable at: http://www.doc.ic.ac.uk/ rak/
14. C. Glymour. *The Mind's Arrows: Bayes Nets and Graphical Causal Models in Psychology*. MIT Press, 2001
15. J. Pearl. *Causality: Models, Reasoning, and Inference.* Cambridge U.P., 2000.
16. L. M. Pereira, H. T. Anh. *Evolution Prospection*, in: K. Nakamatsu (ed.), Procs. First KES Intl. Symposium on Intelligent Decision Technologies (KES-IDT'09), Springer Verlag book in Engineering Series, Himeji, Japan, April 2009.
17. J. J. Alferes, A. Brogi, J. A. Leite, L.M. Pereira. *Evolving logic programs.* Procs. 8th European Conf. on Logics in Artificial Intelligence (JELIA'02), pages 50−61, 2002.
18. 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.
19. 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, September/October 2000.
20. I. Niemelã and 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, LNAI 1265, pages 420–429, 1997.
21. T. Swift. *Tabling for non-monotonic programming.* Annals of Mathematics and Artificial Intelligence, 25(3–4):201-240, 1999.
22. *The XSB System Version 3.0 Volume 2: Libraries, Interfaces and Packages.* July, 2006.