From AAAI Technical Report SS-96-04. Compilation copyright © 1996, AAAI (www.aaai.org). All rights reserved.

Bayesian Network Construction for Contingent Probabilistic Plan Evaluation

Liem Ngo
Department of Electrical Engineering and Computer Science
University of Wisconsin-Milwaukee
PO Box 784, Milwaukee, WI 53201
liem@cs.uwm.edu

Abstract

Contingent probabilistic plans (CPP) have emerged to be an important tool for planning in uncertain and incomplete environments. While Bayesian networks have been used as an efficient mechanism for probabilistic plan projection, their use for CPPs evaluation has not been discussed. In this paper, we address not only the procedures to construct compact Bayesian networks for CPPs evaluation but also their correctness, an aspect neglected by most current Bayesian network approaches. We try to accomplish that by providing a language which combines extended logic programs and probabilistic logic programs to represent probabilistic and deterministic actions models with indirect effects and domain constraints. We propose the concept of Bayesian network-trees which are compact tree structures of Bayesian network fragments and use them to evaluate the probability that a CPP achieves a given goal. We discuss Bayesian network abstraction techniques which can be used to speed up the evaluation process. Our Bayesian network-tree construction algorithm can be proved to be sound and complete, under certain conditions, in our proposed semantics.

Introduction

Bayesian networks (BN) (Pearl 1988) have become the most popular method for representing and reasoning with probabilistic information. There has recently been increasing interest in using BNs to perform plan projection (Kushmerick *et al.* 1995; Darwiche and Goldszmidt 1994; Davidson and Fehling 1994) as well as plan generation (Blythe 1994). Although the use of BNs as the underlying representation provides a uniform and relatively compact representation of both actions and domain relations, the process of computing posterior probabilities (inference) in BNs remains NP-hard (Cooper 1990). This complexity becomes particularly problematic in the large models that arise in plan evaluation. But typically only a small portion of a domain model will be relevant to evaluating any given plan. Recent work has taken advantage of this fact by constructing relatively small networks for evaluating plans from knowledge bases (KB) of network fragments (Blythe 1994; Davidson and Fehling 1994; Lehner *et al.* 1994). But none of this work has provided a formal semantics for the knowledge base from which the networks are constructed. Such a formal semantics is necessary to be able to prove the plan evaluation process correct.

The need for information gathering actions and contingent (or conditional) plans have been well recognized by planning, decision analysis and robotics research communities. A contingent probabilistic planner, C-BURIDAN (Draper *et al.* 1994), has been proposed. C-BURIDAN does not use BNs to perform plan projection. **Contingent probabilistic plans (CPP)** are also the main topic of decision-theoretic planning, either in the form of Influence Diagrams (Howard and Matheson 1984; Pearl 1988) or Markov processes (Dean and Wellman 1991; Boutilier *et al.* 1995b). However, none of this work addresses the plan projection problem by constructing BNs from a KB of action models with indirect effects and domain constraints, e.g. (Baral 1995; Lin 1995), in a rigorous way.

In this paper we provide a comprehensive logical framework for the **representation and evaluation of CPPs**. We aim at the following contributions:

- **A logical representation framework** for probabilistic temporal reasoning and planning is provided. We propose a combination of extended logic programs (Gelfond and Lifschitz 1991) and probabilistic logic programs (Ngo and Haddawy 1995) with well-defined semantics as a means to represent action effects, indirect effects and domain constraints. Extended logic programs have been used as the mainstream logic programming tool for reasoning about deterministic actions and change (Gelfond and Lifschitz 1991). The probabilistic logic programs in (Ngo and Haddawy 1995) have a direct relationship with BNs, which are the main tool for reasoning about probabilistic actions and change (Pearl 1995; Blythe 1994; Davidson and Fehling 1994; Lehner *et al.* 1994). We show that the provided framework is capable of representing all the action models

used in probabilistic planners and in a wide range of decision-theoretic planners.

- **The use of BNs for CPP projection** and a **knowledge-based BN construction for CPP projection** procedure, whose correctness and completeness can be proved, are proposed. We propose the concept of **Bayesian Network-tree (BN-tree)** and use it to encode both the contingencies and the effects caused by a CPP. The available probability updating procedures can be utilized on the constructed BN-tree.

  Essentially, in our framework, deterministic knowledge is used to reason about deterministic effects of actions and as context to index the relevant probabilistic information. This is particularly useful in plan evaluation since when evaluating a plan, the performance of one's own actions is deterministic knowledge, i.e., we know whether or not we plan to attempt an action. So actions can be treated as context to index the portions of the probabilistic knowledge necessary to model their effects. By constructing a BN-tree tailored to specific goals, the performance is improved.

- We show the need of abstraction and propose the use of **local BN abstraction techniques** for CPP evaluation. Some techniques for BN abstraction were presented in (Lam 1994). Unfortunately, they are unsound. In this paper we show that some local transformation rules can be helpful in evaluating dynamic BNs for temporal reasoning.

We use a running toy example, which is inspired by the one in (Davidson and Fehling 1994), to illustrate our representation framework. More details about the encoding of this example in our framework can be found in the full paper. Suppose that a robot is secretly attempting to fetch an object from the floor in an office. It needs to use its gripper to pick up the object and put it in the bag it is carrying. The gripper may be dirty which increases the chance that the robot will drop the object it is holding. We consider three possible actions the robot can perform: use its gripper to pick up an object, put the object held in the gripper in the bag, and clean its gripper. In order to introduce a sensing action, we assume that before cleaning its gripper, the robot should check whether the gripper is currently clean or not because cleaning a clean gripper may make it dirty (or other penalties like delayed time). The robot must avoid detection by a sound sensor which can activate an alarm. We assume that the object location can only be either on the floor, in the robot's gripper or in the bag it carries; and at the beginning the object is on the floor. In order for the robot to plan in this domain, it must be provided with a knowledge base of action model and domain model.

Figure 1.a shows a possible BN model of the *pickup* action. The nodes represent the variables involved in the performance of *pickup*. Object location in the next stage is affected by the action and probabilistically depends on the object's position and the dirtiness of the gripper before the action is performed. If the *pickup* action causes the object to fall on floor, the sound alarm might be activated. Similar interpretations apply to the BN models of the *cleanGripper* and *putinBag* actions depicted in Figures 1.b and 1.d, respectively. Figure 1.c shows the effect of the information sensing action *checkGripper*. Figure 1.e displays the intrinsic relationship between random variables in one time slice. Figure 1.f depicts the persistence rules: status of random variables in future stages depend on current status, if there is no intervening action.

Because of the space limitation, we sketch the representation language and will not provide its formal semantics. Details on those issues can be found in (Ngo *et al.* 1995) and in the full paper.

Representation Language

We have two types of predicates: *probabilistic and deterministic predicates.* Some of the predicates are timed predicates. A timed predicate always has the first attribute indicating the time the associated event or relationship denoted by the predicate occurs. We model only discrete time and throughout the paper we represent the set of time points by the set of integers. If A is a ground timed atom and the time attribute is t, we say A (happens) at time t.

Deterministic predicates (d-predicates) have value true or false and are deterministic. They are used to describe the context the agent is in and to eliminate unnecessary probabilistic information from consideration by the agent. *Deterministic atoms (d-atoms)* and *literals (d-literals)* are atoms and literals formed from d-predicates. A *deterministic base* (DB) is an extended logic program (Gelfond and Lifschitz 1991; Baral and Gelfond 1994), which is a set of universally quantified sentences of the form $L_0 \leftarrow L_1, \ldots, L_m,\ not\ L_{m+1}, \ldots,\ not\ L_n$; where $m, n \geq 0$ and the L_i's are d-literals.

A probabilistic predicate (p-predicate) represents a class of similar random variables. p-predicates appear in probabilistic sentences and are the focus of probabilistic inference processes. An atom formed from a probabilistic predicate is called a *probabilistic atom (p-atom).* Normally, each random variable in a probability model can take values in a finite set and in each possible realization of the world, that variable can have one and only one value. We capture that property by requiring that each p-predicate has at least one attribute. The last attribute of a p-predicate represents the *value* of the corresponding variable. For example, the variable *alarm* can have the value *yes* or *no* and can be represented by a two-position predicate -the first position indicates the time and the second indicates the status of the alarm. We must specify the range of the values of random variables represented by a p-

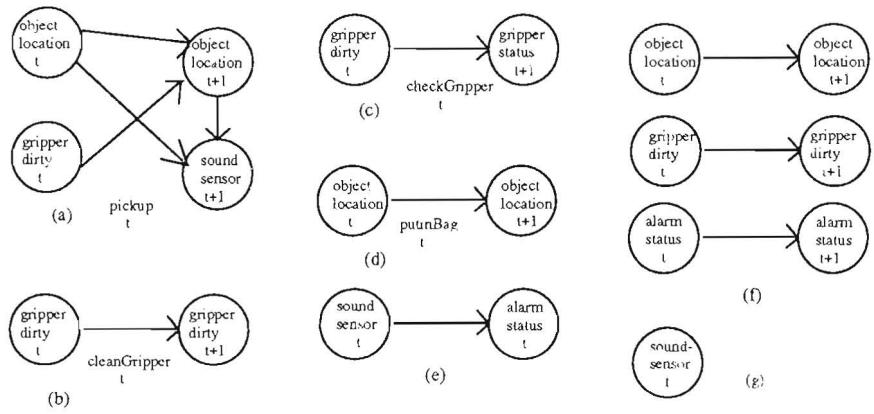

Figure 1: BN models of (a) pickup action, (b) cleanGripper action, (c) checkGripper action, (d) putinBag action, (e) intrinsic causal relationships, (f) persistence relationships and (g) the uncertain status of sound sensor caused by exogenous events.

predicate, so associated with each p-predicate p must be a statement of the form $VAL(p) = \{v_1, \ldots, v_n\}$, where $v_1, \ldots, v_n$ are constants. We denote the set of all such **predicate declarations** in a knowledge base by PD.

Example 1 *The following are declarations of p-predicates in our robot example.* $PD = \{location(T : Time, O : Object, V), VAL(location) = \{floor, inGripper, inBag\}; alarm(T : Time, V), VAL(alarm) = \{yes, no\}; sensor(T : Time, V), VAL(sensor) = \{activated, deactivated\}; gripperdirty(T : Time, V), VAL(gripperdirty) = \{yes, no\}; gripperstatus(T : Time, V), VAL(gripperstatus) = \{clean, dirty\}\}$. *We assume that the domain Object contains only one object named* o_1. *The status report returned by* checkGripper *may be clean (gripper is clean) or dirty (gripper is dirty).*

We interpret the declaration $VAL(alarm) = \{yes, no\}$ as the set of integrity constraints $\{alarm(T, V) \rightarrow V = yes \vee V = no; \leftarrow alarm(T, yes) \wedge alarm(T, no)\}$. Hence, we can represent explicit negative facts on p-atoms. The necessity of explicit negative facts have been well recognized (Gelfond and Lifschitz 1991).

A **probabilistic sentence (p-sentence)** has the form $(P(A_0|A_1, \ldots, A_n) = [\underline{\alpha}, \overline{\alpha}]) \leftarrow L_1, \ldots, L_m,$ *not* $L_{m+1}, \ldots,$ *not* L_p where $n, m, p \geq 0, 0 \leq \underline{\alpha} \leq \overline{\alpha} \leq 1, A_i$ are p-atoms, and L_j's are d-literals. If the above p-sentence is ground then its meaning is "in the context that $L_1, \ldots, L_m$ are believed to be true and none of $L_{m+1}, \ldots, L_p$ is believed to be true then $P(A_0|A_1, \ldots, A_n)$ is in the interval $[\underline{\alpha}, \overline{\alpha}]$. For convenience, we write α instead of the interval $[\alpha, \alpha]$.

We represent a planning framework by a knowledge base KB, which consists of the following components: $KB = \langle PD, PB, DB, CR \rangle$. We have introduced the set of predicate declarations PD. PB and DB contain the action model and the domain knowledge and CR is the set of combining rules used as rules for temporal projection. They will be discussed next.

Representing Planning Problems

Action Representation

Actions are represented by special timed d-predicates which are called **action predicates**. Actions are grouped into classes. The actions in a class differ in action parameters and are called **instances** of the action class. For example, the class of actions *move* can have many instances: each describes a specific action at a specific time, handles a specific object and involves specific source and destination locations. For each action class, we introduce a new timed action predicate and always assume that the name of the predicate is exactly the name of the action class. The first attribute denotes the time when the action occurs and the other possible attributes denote the parameters of the action. For example, the predicate $move(.,.,.,.)$ represents the action class *move* . The atom $move(5, cup_1, loc_A, loc_B)$ represents the action instance: *move, at time 5, the cup* cup_1 *at position* loc_A *to position* loc_B . Henceforth, we use the term action to refer to action instances. We denote by $move(cup, loc_A, loc_B)$ the **generic action:** *move the cup at position* loc_A *to position* loc_B. If A is a generic action, A_t is the action instance *"A is performed at time t"*.

Deterministic effects of actions are represented by clauses in the extended logic program DB (Gelfond and Lifschitz 1991). Most current logic programming solutions to reasoning about actions and change are formulated in extended logic programs, e.g. (Gelfond

and Lifschitz 1993; Baral and Gelfond 1993). Our framework can be seen as an extension of extended logic programs to reasoning about both deterministic and probabilistic actions and change. In this paper, we concentrate on uncertain (probabilistic) effects of actions.

Certain conditions may determine whether an action can be executed; we call these *executability conditions* (or preconditions). If an action is executed, certain conditions may determine the effects that the action will have; we call these *ramification conditions* (Haddawy 1994).

We divide the executability and ramification conditions into two types: uncertain and deterministic. For example, the executability conditions of the *pickup* action might be *the gripper is available* and *the object is on floor*. In our planning scenario we can assume that the information about the availability of the gripper is deterministic and the information about the location of the object is uncertain. Deterministic executability and ramification conditions are represented by d-literals and the uncertain executability and ramification conditions are represented by p-atoms.

We describe each uncertain effect of action(s) in our language by a p-sentence of the form $P(p(T_1,\ldots)|\ldots,q(\ldots),\ldots) = [\underline{\alpha},\overline{\alpha}] \leftarrow a(T_2,\ldots),\ldots,L_r,\ldots,$ *not* L_s where $a(\ldots)$ is an action atom[1], p and q are p-predicates, and the L_i's are d-literals. q and the L_i's may not be timed, but p and a must be timed. If the above sentence is ground, we call $p(T_1,\ldots)$ a *consequent* of $a(T_2,\ldots)$, each $q(\ldots)$ a *(probabilistic) condition* of it, and the conjunction $\ldots \wedge L_r \wedge \ldots \wedge$ *not* L_s the *context* of its corresponding effect.

An action model is a set of p-sentences, each of which has the context including an action atom. In example 2, we present the action model of *cleanGripper* and *checkGripper*. The *checkGripper* action uses a noisy sensor: the reported status may not absolutely correct. Its representation conforms to the common representation of information gathering actions (Pearl 1988; Nicholson and Brady 1994).

Example 2

$P(gripperdirty(T+1,no)|gripperdirty(T,no))$
$\quad = .9 \leftarrow cleanGripper(T);$
$P(gripperdirty(T+1,yes)|gripperdirty(T,yes))$
$\quad = .9 \leftarrow cleanGripper(T);$
$P(gripperstatus(T+1,clean)|gripperdirty(T,no))$
$\quad = .9 \leftarrow checkGripper(T);$
$P(gripperstatus(T+1,dirty)|gripperdirty(T,yes))$
$\quad = .9 \leftarrow checkGripper(T).$

In the full paper, we also show how to represent graphical action models (Kushmerick *et al.* 1995; Haddawy and Suwandi 1994) in our framework.

[1] We can represent the combined effects of multiple actions, such as concurrent actions, by including multiple action atoms in the context.

In this paper, we confine ourselves to an important class of action models, the *Markovian* models (Boutilier *et al.* 1995a). Markov processes can be represented in our language by **Markovian dt-normal action models**.

Definition 1 *An action model is* Markovian dt-normal *or simply* Markovian, *if it can be represented as a finite set of sentences of the form* $Pr(p(T+1,\ldots)|\ldots,q(T_1,\ldots),\ldots) = [\underline{\alpha},\overline{\alpha}] \leftarrow a(T,\ldots),\ldots,L_r,\ldots,$ *not* $L_s,\ldots$ *where* a *is an action predicate. If* $q(T_1,\ldots)$ *is a timed condition then* T_1 *is either* T *or* $T+1$. *All timed d-atoms in the context part of the p-sentence must be at time* T.

Notice that we define the Markovian property on individual random variables rather than on states as is more common (Howard 1971). This helps to substantially reduce the computational complexity of reasoning with the models (Boutilier *et al.* 1995a). Furthermore, our sentence-form representation provides a compact representation of asymmetric and parse link matrices, in the same way as decision tree reresentation of (Boutilier *et al.* 1995b).

Representing Domain Knowledge

Thus far we have only discussed how to describe individual actions but interaction between actions and other causal relationships between variables in a specific domain are important for projecting the consequences of a plan. At least two kinds of domain knowledge are of interest in plan evaluation: intrinsic causal or correlate relationships in the domain and persistence rules. The relationship between *sound sensor* and *alarm* is an intrinsic causal relationship which is not caused by any action of the robot.

Persistence rules are an important component of temporal reasoning. The basic assumption of persistence is that if the state of a variable is not known to be affected by actions or other events over a period of time, it will tend to remain unchanged over that period. In our example, we should have persistence rules for *object location, cleanness of the gripper, sound sensor*, and *alarm*.

Example 3 *A possible set of sentences for the persistence of* location, *and* gripperdirty *is:*

$P(location(T+1,O,floor)|location(T,O,floor)) = 1$
$P(location(T+1,O,floor)|location(T,O,inGripper))$
$\quad = .01$
$P(location(T+1,O,inGripper)$
$\quad |location(T,O,inGripper)) = .99$
$P(gripperdirty(T+1,yes)|gripperdirty(T,yes)) = 1$
$P(gripperdirty(T+1,no)|gripperdirty(T,no)) = 1$

The uncertainty in the future location of the object when it is held in the robot's gripper is due to the uncertain **exogenous events**.

In most probabilistic (Kushmerick *et al.* 1995) and decision-theoretic (Haddawy and Suwandi 1994) planning systems, a planning problem is characterized by

an initial state probability distribution. We represent such a distribution by a set of prior probability sentences at time 0. In the robot example, the effect of **exogenous events** on the sound sensor is represented by a prior probability distribution $sensor(T, .)$ at all time points.

The Combining Rules

When two or more actions or events influence the state of a variable, we need to know the probability distribution of that variable given all possible combinations of states of the actions and events. For example, if B and C influence A we need to know $P(A|B, C)$. This information can be specified explicitly but it is often difficult to obtain. Typically we will only have the probabiltiy of a variable conditioned on each of the individual influences, e.g. $P(A|B)$ and $P(A|C)$. In such situations we need some way of inferring the combined influences from the individual influences. Combining rules such as generalized noisy-OR and noisy-AND (Srinivas 1993) are commonly used to construct similar combined influences in BNs.

The topic of temporal reasoning about actions with indirect effects and domain constraints has been attracting much attention amongst researchers who focus on classical logic solutions (Lin 1995; Giunchiglia *et al.* 1995; Baral 1995: Kartha and Lifschitz 1994; McCain and Turner 1995) as well as researchers who focus on BN-approaches (Pearl 1995; Blythe 1994; Davidson and Fehling 1994). In (Lin 1995; McCain and Turner 1995), the concept of causality is emphasized and used as the main tool to derive the ramifications of performing an action. Combining rules are used as default rule for probabilistic reasoning about inter-causal influences (Srinivas 1993). In (Giunchiglia *et al.* 1995; Kartha and Lifschitz 1994), the ramification problem is solved by classfying the fluents into inertial and dependent. The state-transition semantics is based, intuitively, on that classification and the causes of the change. In our model we incorporate such manipulations into some procedures—the combining rules. A combining rule takes as input *a set of p-sentences* which have the same random variable in the consequents—the causes that influence the random variable—and produces the combined effect.

We define *a combining rule* as any algorithm that takes as input a set of *ground* probabilistic sentences with the same consequent $\{P(A_0|A_{i1}, \ldots, A_{in_i}) = [\underline{\alpha}_i, \overline{\alpha}_i] \leftarrow \ldots | 1 \leq i \leq m\}$ and produces as output $P(A_0|A_1, \ldots, A_n) = [\underline{\alpha}, \overline{\alpha}]$, where $A_1, \ldots,$ and A_n are all different and $\{A_1, \ldots, A_n\}$ is a subset of $\cup_{i=1}^{m}\{A_{i1}, \ldots, A_{in_i}\}$.

The combining rules are generally dependent on the concrete domain. One plausible rule is actions take precedence over other causes. It is illustrated in the following example:

Example 4 *Suppose that the pickup action is chosen to be performed at time 5 on object o_1. The combined effect of the pickup action and the persistence rule for plan evaluation might contain the following sentences (to shorten the example, we introduce variables into the sentences):*

$$P(location(6, o_1, inGripper) \mid location(5, o_1, floor), gripperdirty(no)) = .85$$
$$P(location(6, o_1, floor) \mid location(5, o_1, floor), gripperdirty(no)) = .15$$
$$P(location(6, o_1, inGripper) \mid location(5, o_1, floor), gripperdirty(no)) = .90$$
$$\ldots$$
$$P(location(6, o_1, inGripper) \mid location(5, o_1, inGripper), gripperdirty(Z)) = .99$$
$$P(location(6, o_1, floor) \mid location(5, o_1, inGripper), gripperdirty(Z)) = .01$$
$$\ldots$$

The first set of sentences is constructed for the case the conditions of pickup are satisfied. The probability values are taken from the action models. When the object is in the gripper at time 5, the action is not performed and the probability values are inherited from persistence rules.

The status of alarm is an example of combining indirect effect $(sensor(t) \rightarrow alarm(t))$ and persistence $(alarm(t-1) \rightarrow alarm(t))$.

Instead of a *successor state axiom* for each *fluent* (Reiter 1991), the KB needs to contain at least one combining rule for each p-predicate.

The Planning Framework

For a planning domain, we can construct a knowledge base KB, which consists of the following components: $KB = \langle PD, PB, DB, CR \rangle$. PD is the set of predicate declarations. PB contains the action model and the domain knowledge—intrinsic causal relationships, persistence rules and other probabilistic information. DB is the deterministic base and CR is the set of combining rules. We say that a KB is Markovian dt-normal if PB is.

Contingent Probabilistic Plans

We assume that the underlying KB is Markovian and KB does not contain any fact about a constant time point other than the starting time 0. We represent a CPP using a programming language in which the primitive statements are generic actions and there are only two control structures: sequential and conditional (by using the CASE construct). The condition of CASE can refer to the values of RVs in previous time slices. We always assume that in a CASE construct the different branching conditions are mutually exclusive.

Example 5 *Consider the following CPP for the robot:*

checkGripper;
CASE *gripperstatus* *IS*
dirty : *cleanGripper*;
ENDCASE;
pickup(o_1);
putinBag(o_1).

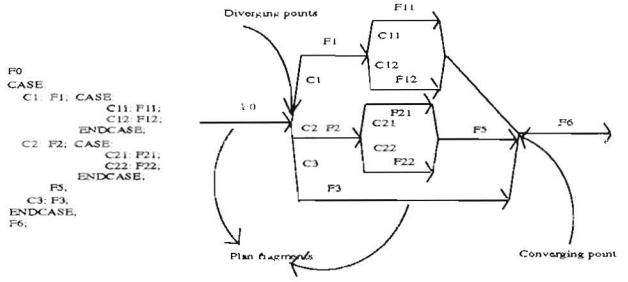

Figure 2: The graph model of a contingent probabilistic plan.

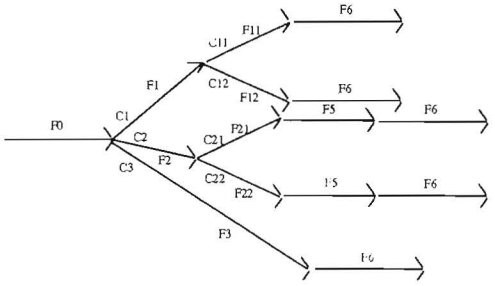

Figure 3: The tree model of a contingent probabilistic plan.

The action cleanGripper *is performed only when the gripper is observed dirty.*

In a CPP, we call a sequence of consecutive generic actions in the plan which does not contain the CASE or ENDCASE keywords a (sequential) plan fragment. A CPP can be represented by a graph of its maximal plan fragments as shown in the figure 2. In the figure, the names starting with F denote a maximal plan fragment. In the graph representation, each horizontal bar represents a maximal plan fragment and is annotated with the corresponding name of the fragment. The lines connecting the horizontal bars represent the diverging (corresponding to the CASE keyword) and converging (corresponding to the ENDCASE keyword) links. The diverging links are annotated with the corresponding conditions in the program. Equivalent to the graph representation is the tree of maximal plan fragments, shown in Figure 3.

The Goal of Plan Projection

In this paper we are interested in evaluating the probability of some random variables, which are called **goal variables**, at the end of the performance of a CPP. Notice that such a plan has several branching possibilities, each branch has a specific time length and a specific probability of occurrence. Assume the CPP $\mathbf{P}$ has n possible branches $\mathcal{F}_i, i = 1, \ldots, n$, the probability of occurrence of branch $\mathcal{F}_i$ is $Pr_{branch}(i)$, the length (duration) of branch $\mathcal{F}_i$ is n_i and we want to evaluate the probability that an (atemporal) random variable X achieving the value x. The desired probability is given by the following formula:

$$Pr^*(X = x|\mathbf{P}) = \sum_{i=1}^{n}(Pr(A_i|\mathcal{F}_i) \times Pr_{branch}(i))$$

where A_i is the ground p-atom in our language representing the fact that the random variable X achieving the value x at time n_i and $Pr(A_i|\mathcal{F}_i)$ is the probability of A_i when $\mathcal{F}_i$ is actually performed.

Example 6 *In the example 5,* $X = x$ *may be interpreted as "*o_1 is in the bag". $\mathcal{F}_1$ *is* $\langle checkGripper(0); cleanGripper(1); pickup(2, o_1); putinBag(3, o_1)\rangle$, $n_1 = 4$, $A_1 = location(4, o_1, inBag)$, $Pr(A_1|\mathcal{F}_1)$ *is the probability of* A_1 *after* $\mathcal{F}_1$ *is actually performed and* $Pr_{branch}(1)$ *is the probability that* checkGripper *returns dirty. Similarly,* $\mathcal{F}_2$ *is* $\langle checkGripper(0); pickup(1, o_1); putinBag(2, o_1)\rangle$, $n_2 = 3$, $A_2 = location(3, o_1, inBag)$, $Pr(A_2|\mathcal{F}_2)$ *is the probability of* A_2 *after* $\mathcal{F}_2$ *is performed and* $Pr_{branch}(2)$ *is the probability that* checkGripper *returns clean.*

Notice that our procedures can be used to construct BNs for evaluating RVs at any time point. Hence, they can be embedded in decision-theoretic planners which produce optimal policies.

Plan Fragments and Their BN Models

A (sequential) plan fragment is a sequence of *generic actions*. If F is the plan fragment $\langle A^{(1)}, \ldots, A^{(n)}\rangle$ and t_0 is a time point, we use F_{t_0} to denote the "concrete" plan $\langle A^{(1)}_{t_0}, \ldots, A^{(n)}_{t_0+n-1}\rangle$. Notice that our KB is Markovian, and so, each action takes one unit amount of time. We say t_0 is the *starting time* and $t_0 + n$ is the *ending time* of F_{t_0}.

In order to build a BN model to evaluate a CPP, we construct the BN model for each maximal plan fragment. The BN model of the CPP will be a tree of those component BNs. BN-trees will be introduced in the next section.

Assume we want to construct a BN to evaluate the effect of a plan fragment F on a set of random variables. We assume that F is performed at a time point t_0, where t_0 is a variable, and try to construct a parameterized BN for F which will be instantiated to concrete BN at concrete time points when we combine the fragments. If KB does not contain any fact related to any constant time point except time point 0 (the starting state) and KB is Markovian then we formulate the problem as evaluate the effect of the "concrete" plan F_{t_0} on the goal random variables at time $t_0 + n$.

In the full paper, we adapt a procedure which was presented in (Ngo *et al.* 1995) to the current framework to construct *conditional BNs*.

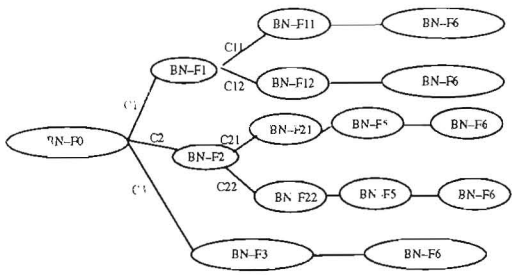

Figure 5: A BN-tree model of a contingent probabilistic plan.

The concept of *conditional Bayesian networks* has been used by several researchers, including (Davidson and Fehling 1994). We call a DAG a conditional BN if we can construct from it a BN (with the corresponding probability independence assumptions) by assigning (prior) probability distributions to some of its nodes which do not have incoming arcs. We call such nodes *input variables*. In our planning settings the input variables are variables at the starting time point.

We denote the BN constructed by our procedure when the input consists of the plan fragment F, the starting time t_0, ending time t_0+n and the set of goal random variables G by $BN(F, t_0, t_0+n, G)$.

The figure 4 shows three conditional BNs constructed by our procedure for three plan fragments. The purpose is constructing the BN relevant to the evaluation of the final goal variables $location(o_1)$ and $alarm$ after performing $\langle pickup(o_1), putinBag(o_1)\rangle$. The process starts with the final goal variables $location(o_1)$ and $alarm$ and the action $putinBag(o_1)$. The input variables of $BN(\langle putinBag(o_1)\rangle, t_0+1, t_0+2, \{location(t_0+2, o_1), alarm(t_0+2)\})$ become the goal variables of $BN(\langle pickup(o_1)\rangle, t_0, t_0+1, \{location(t_0+1, o_1), alarm(t_0+1)\})$. Notice that irrelevant RVs, e.g. $gripperdirty(t_0+1)$, are not considered.

Bayesian Network-Trees

A CPP can be represented as a tree in which each link is annotated with an optional condition and a plan fragment F (see Figure 3). In the previous section we presented the concept of $BN(F, t_0, t_0+n, G)$, the BN for evaluating the effect of F performed at t_0 on the goal random variables G at t_0+n. In order to evaluate the whole CPP, we connect those BNs into a BN-tree.

Example 7 *If the CPP is represented as a tree of maximal plan fragments in Figure 3 and we have a BN for each maximal plan fragment then we can connect those BNs into a tree form shown in Figure 5. In the Figure 5, each BN-F_i is a conditional BN of F_i.*

In this section we formally define the concept of BN-tree. We believe that the concept of BN-tree is helpful in a variety of problems. To avoid the confusion between the nodes, edges, paths, etc. of the BN-tree and those of each BN in the tree, we call a node of the BN-tree a T-node,etc.

Definition 2 *A BN-tree is a tree in which:*

(1) Annotated to each T-node is a tuple $\langle BN_i, S_i\rangle$, where BN_i is a conditional BN and S_i is a set of some root nodes (nodes without preceding nodes) in BN_i. We require that:

(1.1) There exists a link matrix for each node in BN_i which is not in S_i. The link matrices of nodes in S_i are provided by BNs at higher level in the BN-tree.

(1.2) For any T-branch, any node in the BNs of that branch is defined once and only once (i.e. has one and only one link matrix).

(2) Annotated to each T-edge is the context of the T-subtree *originated from the T-node at the end of the T-edge. A context is a disjuntion of conjunctions of the form $\bigvee_{i=1}^{m}\bigwedge_{j=1}^{n} val(N_{ij}, v_{ij})$, where $val(N_{ij}, v_{ij})$ means the node N_{ij} achieves the value v_{ij}. We require that:*

(2.1) Each conjunction $\bigwedge_{j=1}^{n} val(N_{ij}, v_{ij})$ is consistent.

(2.2) Each node N_{ij} appears in at least one of the BNs in the ancestor T-nodes of the T-edge.

(2.3) For any T-branch, the conjunction of all contexts on the T-edges is consistent.

(2.4) For every non-leaf T-node on the BN-tree, the contexts of any two different T-edges departing from that T-node are mutually inconsistent.

A T-branch is any (directed) T-path from the T-root of the BN-tree to a T-leave.

Each set S_i is the set of input variables of the corresponding BN in our planning setting.

A BN-tree is *complete* if (1) The set of contexts annotated to the T-edges departing from an arbitrary T-node of the BN-tree is covering. (2) The sets of nodes (random variables) in the BNs along any two T-paths from the T-root to a T-leaf are the same.

Probability Distribution Implied by a BN-tree

Let b be a T-branch. The *BN implied by b*, denoted by BN_b, is constructed by connecting all the BNs at the T-nodes along b. That BN is well defined. In our planning setting each BN_b is the BN model of a branch $\mathcal{F}_i$ (see section).

Definition 3 *Let N be the set of all nodes (random variables) appearing in a complete BN-tree T. The* probability distribution implied by T, *denoted by Pr_T, is defined as:*

$$Pr_T(X) = Pr_b(X),$$

where X is an arbitrary assignment to all random variables in N, b is the only T-branch whose conditions are compatible with X, and Pr_b is the probability distribution of BN_b.

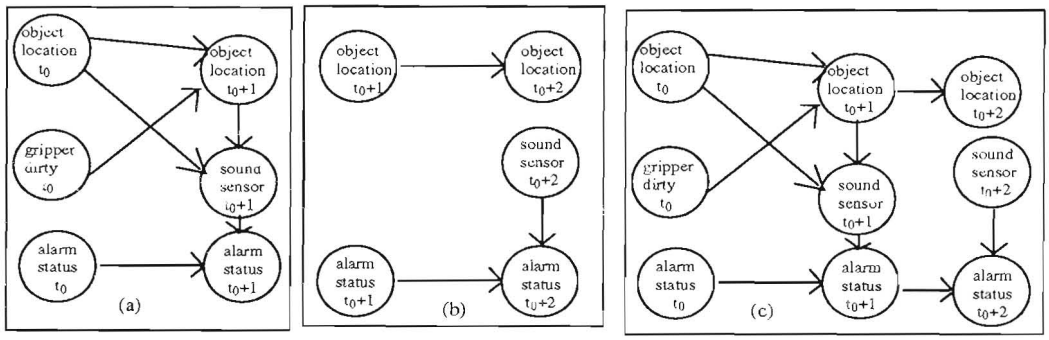

Figure 4: (a) $BN(\langle pickup(o_1)\rangle, t_0, t_0+1, \{location(t_0+1, o_1), alarm(t_0+1)\})$; (b) $BN(\langle putinBag(o_1)\rangle, t_0+1, t_0+2, \{location(t_0+2, o_1), alarm(t_0+2)\})$; and (c) $BN(\langle pickup(o_1), putinBag(o_1)\rangle, t_0, t_0+2, \{location(t_0+2, o_1), alarm(t_0+2)\})$.

Proposition 1 *If T is a complete BN-tree then Pr_T is a well-defined probability function.*

In case T is not complete, we can use some default rules to derive a default probability distribution. In one default rule we assume that if a RV does not appear in a BN_{Branch} then it will takes a predefined value and be probabilistically independent to other RVs.

A straightforward procedure to update probability of some RVs G on a BN-tree upon receiving the evidence E, e.g. $Pr_T(G|E)$, is

(1) Select from T the set B of T-branches whose contexts are compatible with $G \cup E$.

(2) If the context of a branch b is C_b then compute $Pr_b(G, C_b|E)$

(3) $Pr_T(G|E) = \sum_b Pr_{b\in B}(G, C_b|E)$.

We are currently investigating inference procedures on BN-trees.

Bayesian Network Abstraction

Abstract Bayesian Networks

We can see in Figure 3 that a plan fragment may be repeated many times in a CPP. We can speed up the reasoning process by abstracting away the "unnecessary" nodes in the BN representing a plan fragment and storing the abstract BN. In the abstraction process we need to retain the interface and evidence variables. The interface variables consist of *input variables* which form the interface with preceeding BNs and the *output variables*, included in this class are goal variables, which form the interface with the succeeding BNs or for querying purpose. The evidence variables are used for probability updating. For our current purposes, they are *observation variables* whose values are used in later branching conditions. *Internal variables* are the variables which are not classified as interface or evidence. They can be abstracted away without affecting the reasoning on the whole BN-tree.

Definition 4 *An* **abstract BN** *of a conditional BN $\mathcal{N}$ is a conditional network $\mathcal{M}$ such that: (1) $\mathcal{N}$ and $\mathcal{M}$ have the same interface and evidence variables.*

(2) Any node in $\mathcal{N}$ is in $\mathcal{M}$.

(3) For any probability distribution assigned to the input variables, the resulting marginal probability distribution of the random variables common to the two networks are the same.

In (Lam 1994) the author proposes a technique to abstract away a portion of a BN. That technique turns out to be unsound: the resulting network does not represent the marginalized probability distribution of the original probability distribution. We propose a less extensive but sound approach of abstraction. For example, the transformation in Figures 6.(a) is useful in temporal reasoning. The sequence of nodes $(1) \rightarrow (2) \rightarrow \ldots(n)$ can occur because of the persistence rules. Assume node i has m_i values. The clustering algorithm would create $n-1$ clusters of size $m_0 \times m_i \times m_{i+1}, i = 1, \ldots, n-1$. The network after transformation has only one cluster of size $m_0 \times m_1 \times m_n$. In most cases, the new network is more efficient than the original. In Figure 6.(a) we assume that nodes $(2), \ldots, (n-1)$ are internal nodes, there are no arrows connected to them other than the ones shown in the figure and node (n) has only two incoming arcs. The new link matrix of node (n) contains $Pr(x_n|x_1, x_0)$, where x_i is the RV associated with node (i).

Two other transformations are given in Figures 6.(b) and (c). In Figure 6.(b) we assume that there are no arrows connected to (2) other than the ones shown in the figure. In order to use the transformation in Figure 6.(c), we require that the arc departing from (0) in the figure is the only one connected to (0). We also can use some transformations, which are described by Shachter (Shachter 1988), for abstraction purpose. In the full paper, we prove the soundness of our abstraction rules.

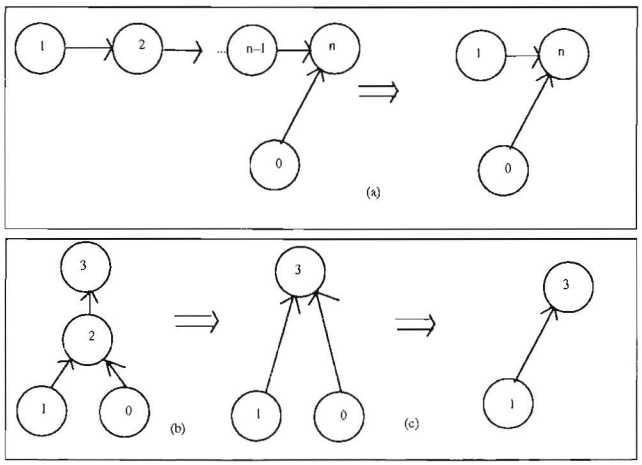

Figure 6: Two local transformations on BNs.

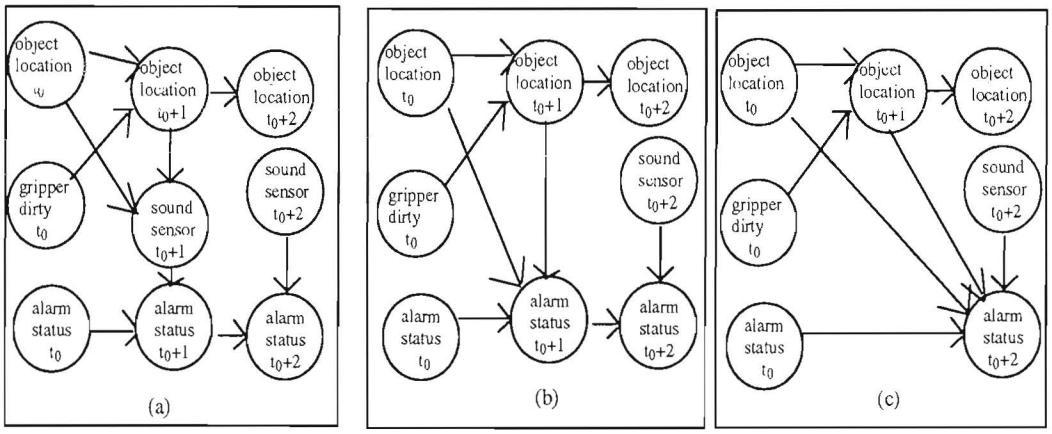

Figure 7: (a) $BN(\langle pickup(o_1), putinBag(o_1)\rangle, t_0, t_0+2, \{location(t_0+2, o_1), alarm(t_0+2)\})$. (b) The BN after $sensor(t_0+1)$ is abstracted away. (c) The BN after $alarm(t_0+1)$ is abstracted away.

Figure 7 shows the application of abstraction techniques in our planning example.

Abstract Junction Trees

The most efficient technique to update a BN in light of evidence builds a *junction tree* from the network and do the probability propagation on it. To save computation time, we may want to store either the abtract BN or the abstract junction tree associated with a plan fragment for later use.

Assume we have a conditional BN. We construct the abstract junction tree by the following steps:

(1) Performing the local transformations to simplify the network;

(2) Connecting each pair of output variables by an undirected edge;

(3) Connecting each pair of input variables by an undirected edge;

(4) Connecting all parents of an arbitrary node by undirected edges;

(5) Abstracting away the internal nodes, if possible and advantageous;

(6) Assigning to each input variable a Belief Table of 1's;

(7) Triangulating the network and build the junction tree.

The above steps can be proved to preserve the (marginal) joint probability distribution.

Constructing a BN-tree to Evaluate a Contingent Probabilistic Plan

Assume F and F' are two plan fragments and BN and BN' are two (maybe abstract) BNs constructed from F and F' respectively. We say the pair $\langle F, F'\rangle$

is *compatible* if the set of input variables of BN' is a subset of the set of output variables of BN.

Proposition 2 *If any two consecutive BNs in Figure 5 are compatible then the result structure is a BN-tree.*

Figure 8 shows the BN fragments resulted from the CPT in example 5. In Figure 8.(d) is the BN for evaluating the branch $\langle checkGripper(), cleanGripper(), pickup(o_1), putinBag(o_1)\rangle$ on the BN-tree. In that BN, an abstracted version of $BN(\langle putinBag(o_1)\rangle, t_0, t_0 + 1, \{location(t_0 + 1, o_1), alarm(t_0 + 1)\})$ is used.

In the full paper we investigate the construction of a tailored BN-tree for the evaluation of the effect of a CPP on some given random variables. Essentially, the procedure is a backtracking process building the conditional BNs for each fragment starting from last one. In order to assure the compatibility between any two consecutive BNs, in the backtracking process, the input variables of the BNs of all fragments departing from one (diverging or converging) point are used as goal variables in building the BN of the fragment(s) immediately preceeding that point. We have shown the soundness and completeness properties of our procedure in certain conditions.

Notice that we do not generate the whole BN-tree but only the portion relevant to a set of goal variables. That strategy makes the procedure more efficient.

Conclusions and Future Research

In this paper we propose a language which combines extended logic programs and probabilistic logic programs to represent probabilistic action models and domain knowledge for plan projection purpose. The language has a formal semantics, yet inherits the practicality property of logic programming and BNs. We argue that classical logics techniques for temporal reasoning should be translated into combining rules in our probabilistic setting. We propose the concept of BN-Trees and abstraction techniques for evaluating contingent probabilistic plans.

In the full paper, we embed the discussed concepts and techniques in a decision-theoretic planner. We propose a **Knowledge-Based Interactive Decision-Theoretic Planning System**, which can be used as a tool for decision-theoretic plan generation. In such a system, the action model and domain knowledge are stored in a KB. The user controls the process by specifying to the system the utility function and the constraints on the plan space, which are used by the system to find the optimal policies. To specify the constraints on the plan space, we use a technique similar to that of GOLOG (Levesque *et al.* 1995), which is an effort to extend situation calculus (without considering quantitative uncertainty) for robot control problems. In GOLOG, a plan (or complex action, in their terminology) is composed from primitive actions by using conventional programming language constructs (sequential, conditional, looping, etc.). Some primitive actions are nondeterministic, and hence, specifying a set of plans. The GOLOG interpreter, upon receiving a complex action specification, produces the *valid plans* -the plans consist of actions whose preconditions are proved to be satisfied after performing their corresponding preceeding actions. In our system, the plan space is also represented by a program but the objective is finding the plans that maximize the expected utility.

We are currently investigate more efficent procedures for finding optimal plan(s). We are also interested in anytime algorithms which could find "good" plans in resource constraint environments (Horvitz 1989).

Acknowledgement

The author would like to thank his supervisor, Prof. Peter Haddawy, for his various help. This work was supported by a UWM graduate school fellowship and partially by NSF grant IRI-9509165.

References

C. Baral and M. Gelfond. Representing concurrent actions in extended logic programs. In R. Bajcsy, editor, *Proceedings of IJCAI'93*, pages 866–871, Chambery, France, 1993.

C. Baral and M. Gelfond. Logic programming and knowledge representation. *Journal of Logic Programming*, 19(20):73–148, 1994.

C. Baral. Reasoning about actions: Nondeterministic effects, constraints and qualification. In C. Mellish, editor, *Proceedings of IJCAI'95*, Montreal, Canada, 1995.

J.S. Blythe. Planning with external events. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, Seattle, July 1994.

C. Boutilier, Thomas Dean, and Steve Hanks. Planning under uncertainty: Structural assumptions and computational leverage. In *Proceedings of the Third European Workshop on Planning*, 1995.

C. Boutilier, R. Dearden, and M. Goldszmidt. Exploiting structure in policy construction. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Montreal, Canada, August 1995.

G.F. Cooper. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence*, 42(2-3):393–405, 1990.

A. Darwiche and M. Goldszmidt. Action networks: A framework for reasoning about actions and change under uncertainty. In *Proceedings of The Tenth Conference on Uncertainty in AI*, pages 136-144, Seattle, July 1994.

R. Davidson and M.R. Fehling. A structured, probabilistic representation of action. In *Proceedings of The Tenth Conference on Uncertainty in AI*, pages 154–161, Seattle, July 1994.

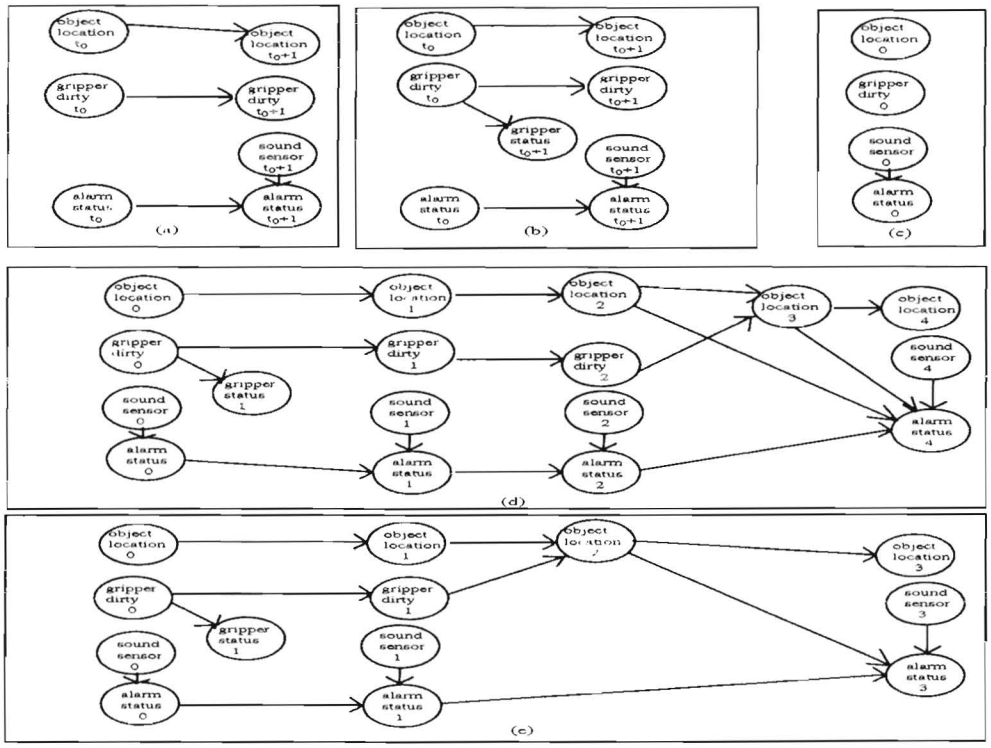

Figure 8: (a) $BN(\langle cleanGripper()\rangle, t_0, t_0+1, \{location(t_0+1, o_1), gripperdirty(t_0+1), alarm(t_0+1)\})$. (b) $BN(\langle checkGripper()\rangle, t_0, t_0+1, \{location(t_0+1, o_1), gripperdirty(t_0+1), gripperstatus(t_0+1), alarm(t_0+1)\})$. (c) $BN(\langle\rangle, 0, 0, \{location(0, o_1), gripperdirty(0), alarm(0)\})$ or the initial state. (d) $BN(\langle checkGripper(), cleanGripper(), pickup(o_1), putinBag(o_1)\rangle, 0, 4, \{location(4, o_1), alarm(4)\})$ using an abstracted version of $BN(\langle putinBag(o_1)\rangle, t_0, t_0+1, \{location(t_0+1, o_1), alarm(t_0+1)\})$. (e) $BN(\langle checkGripper(), pickup(o_1), putinBag(o_1)\rangle, 0, 3, \{location(3, o_1), alarm(3)\})$.

T.L. Dean and M.P. Wellman. *Planning and Control.* Morgan Kaufmann, San Mateo, CA, 1991.

D. Draper, S. Hanks, and D. Weld. Probabilistic planning with information gathering and contingent execution. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, pages 31–36, Chicago, June 1994.

Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, pages 365-385, Sept 1991.

Michael Gelfond and Vladimir Lifschitz. Representing actions and change by logic programs. *Journal of Logic Programming*, (17):301-321, 1993.

E. Giunchiglia, G. N. Kartha, and V. Lifschitz. Actions with indirect effects (extended abstract). In C. Boutilier and M. Goldsmidtz, editors, *Working notes of the AAAI Spring Symposium on Extending Theories of Action: Formal Theories and Practical Applications*, pages 80-85, Stanford, CA, 1995.

P. Haddawy and M. Suwandi. Decision-theoretic refinement planning using inheritance abstraction. In *Proceedings of The Second International Conference on AI Planning Systems*, pages 266-271, June 1994.

P. Haddawy. *Representing Plans Under Uncertainty: A Logic of Time, Chance, and Action*, volume 770 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, 1994.

E.J. Horvitz. Reflection and action under scarce resources: Theoretical principles and empirical study. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1121-1127, Detroit, MI, August 1989.

R. A. Howard and J. E. Matheson. Influence diagrams. In R. A. Howard and J. E. Matheson, editors, *The Principles and Applications of Decision Analysis, Vol II.* Strategic Decisions Group, Menlo Park, CA, USA, 1984.

R. A. Howard. *Dynamic Probabilistic Systems. Vols I and II.* John Wiley and Sons, Inc., 1971.

G. N. Kartha and V. Lifschitz. Actions with indirect effects (preliminary report). In *Proceedings of the Fourth International Conference on Principles of*

Knowledge Representation and Reasoning, pages 341 - 350, 1994.

N. Kushmerick, S. Hanks, and D. Weld. An algorithm for probabilistic planning. *Artificial Intelligence Journal*, 1995.

Wai Lam. Abstraction in bayesian networks. In *The Twelfth National Conference on Artificial Intelligence*, 1994.

P.E. Lehner, C. Elsaesser, and S.A. Musman. Constructing belief networks to evaluate plans. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 416–422, Seattle, July 1994.

H.J. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. B. Scherl. Golog: A logic programming language for dynamic domains. *Journal of Logic Programming, Special Issue on Reasoning about Actions and Change*, 1995. Submitted.

Fangzhen Lin. Embracing causality in specifying the indirect effects of actions. In *Proceedings of the 1995 International Joint Conference on Artificial Intelligence*, Montreal, Canada, August 1995.

N. McCain and H. Turner. A causal theory of ramifications and qualifications. In *Proceedings of the 1995 International Joint Conference on Artificial Intelligence*, Montreal, Canada, August 1995.

Liem Ngo and P. Haddawy. Probabilistic logic programming and bayesian networks. In *Proceedings of Asian Computing Science Conference-95*, Dec 1995.

Liem Ngo, P. Haddawy, and J. Helwig. A theoretical framework for temporal probability model construction with application to plan projection. In P. Besnard and S. Hanks, editors, *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 419- 426, Montreal, Canada, 8 1995.

A.E. Nicholson and J.M. Brady. Dynamic belief networks for discrete monitoring. *IEEE Transactions on Systems, Man and Cybernetics*, 24(11):1593–1610, Nov 1994.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1988.

J. Pearl. Actions as local surgery. In C. Boutilier and M. Goldszmidt, editors, *Working Notes of AAAI Spring Symposium on Extending Theory of Actions*, April 1995.

R. Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359- 380. Academic Press, San Diego, CA, 1991.

R.D. Shachter. Probabilistic inference and influence diagrams. *Operations Research*, 36:589–604, 1988.

S. Srinivas. A generalization of the noisy-or model. In *Proceedings of the Ninth Conference on Uncertainty in AI*, pages 208–217, July 1993.