INTEGRATING REASONING ABOUT ACTIONS AND BAYESIAN NETWORKS

Yves Martin[1] and Michael Thielscher[2]
[1] *SAP AG, SAP Research CEC Dresden, Chemnitzer Strasse 48, 01187 Dresden, Germany, yves.martin@sap.com*
[2] *Artifical Intelligence Institute, Technische Universität Dresden, 01062 Dresden, Germany, mit@inf.tu-dresden.de*

Keywords: Knowledge Representation and Reasoning

Abstract: According to the paradigm of Cognitive Robotics (Reiter, 2001a), intelligent, autonomous agents interacting with an incompletely known world need to reason logically about the effects of their actions and sensor information they acquire over time. In realistic settings, both the effect of actions and sensor data are subject to errors. A cognitive agent can cope with these uncertainties by maintaining probabilistic beliefs about the state of world. In this paper, we show a formalism to represent probabilistic beliefs about states of the world and how these beliefs change in the course of actions. Additionally, we propose an extension to a logic programming framework, the agent programming language FLUX, to actually infer this probabilistic knowledge for agents. Using associated Bayesian networks allows the agents to maintain a single and compact probabilistic knowledge state throughout the execution of an action sequence.

1 Introduction

One of the most challenging and promising goals of Artificial Intelligence research is the design of autonomous agents, including robots, that solve complex tasks in a dynamic world. Achieving a high degree of autonomy in partially known environments requires the high-level cognitive capability of representation and reasoning. In realistic settings, both the effectors and the sensors of agents will be subject to uncertainty. The agent can only hold a degree of belief about the actual state of the world. Subsequent measurements may then help to decrease the degree of uncertainty. In this paper, we introduce a representational formalism —based on the *fluent calculus* (Thielscher, 1999)— and a computational approach to represent and compute the inferences required to update an agent's beliefs about the world in accordance with the effects of the various actions it performs.

In order to represent the probability of possible states of an agent's environment in fluent calculus, in this paper we generalize the axiomatization of knowledge of an agent (Thielscher, 2000) and its beliefs (Jin and Thielscher, 2004), respectively, by introducing a new function *PState*. This function assigns a probability to each state. This is accompanied by a transition probability, which denotes the likelihood of a new state relative to a specific action performed in the preceding state. The probabilistic update from one state to the next can then be defined just like standard Bayesian update. While this approach is similar to (Bacchus et al., 1999), our notion of state probability allows for evaluating the likelihood of state properties directly in the updated state.

To actually infer new knowledge and reason about the execution of actions, we use logic programming in the agent programming language FLUX, which is based on our representation formalism, the fluent calculus. In FLUX, atomic properties of states can likewise be evaluated directly wrt. a so-called FLUX knowledge state. This (incomplete) state represents the set of all states considered possible by the agent. From a computational perspective, this is of course the only practical way as it avoids literally computing with every possible state (Thielscher, 2005a). In this paper, we extend FLUX so as to represent the aforementioned notion of state probability without losing the merits of having to cope with only one knowledge state rather than all possible states. To achieve this, we associate a Bayesian network with such a knowledge state. The relationships of the nodes in such a network denote probabilistic (causal) links between fluents introduced through the execution of actions. Using the contextual variable elimination algorithm (Poole and Zhang, 2003), we can infer and update the probabilities of properties after the execution of (non-)sensing actions. The associated network is updated, too, in order to maintain a compact representation which allows for efficient inferences. In this

way, we only have to encode the (relatively few, in a modular domain) local dependencies between properties of the state in the Bayesian network instead of representing the global state space. This contrasts our method to approaches in (Reiter, 2001a; Grosskreutz and Lakemeyer, 2000), where each possible state is represented by a different sequence of actions, and searching through the resulting state space can lead to intolerable computation times.

The rest of the paper is organized as follows: In the next section, we briefly review the fluent calculus and extend it with the notions of state probability and probabilistic state update axioms. In Section 3, we give a brief introduction to FLUX and then show how to associate a Bayesian network to a FLUX knowledge state and to efficiently infer the updates. A summary and discussion on related and future work concludes the paper in Section 4.

2 Representing Probabilistic Knowledge in the Fluent Calculus

2.1 The basic fluent calculus

Fluent calculus uses atomic properties of states, called fluents, to represent the internal structure of the domain. Fluents are represented as terms of sort FLUENT and, semantically speaking, a state is the collection of all fluents that hold in a situation. Formally, every term of sort FLUENT is also of sort STATE, and if z_1, z_2 are states, then so is $z_1 \circ z_2$. A fluent is defined to hold in a state just in case it is contained in it:

$$Holds(f : \text{FLUENT}, z : \text{STATE}) \stackrel{\text{def}}{=} (\exists z')\, z = f \circ z'$$

The foundational axioms of the fluent calculus (see, e.g., (Thielscher, 2005b)) ensure that, essentially, states are interpreted as non-nested sets of fluents. The term $State(s : \text{SIT})$ denotes the state at a situation, where a situation is the sequence of actions that has been performed by the agent. The special constant S_0 denotes the initial situation and the constructor $Do(a,s)$ then maps an action a and a situation s to the situation after the performance of the action. $State(s : \text{SIT})$ allows to define the expression $Holds(f, s : \text{SIT})$ as an abbreviation for $Holds(f, State(s))$. The basic fluent calculus thus allows to represent incomplete knowledge of a situation by means of standard first-order logic.

The use of states in the fluent calculus allow the fundamental frame problem to be solved on the basis of an axiomatic characterization of two functions, $-$ and $+$, for removal and addition of sub-states. Due to lack of space, we omit the details and just mention the following result of how the fundamental frame problem is solved in the basic fluent calculus (Thielscher, 2005b):

Proposition 1 *Let $z_2 = z_1 - z^- + z^+$, then the foundational axioms of the fluent calculus entail*[1]

$$\begin{aligned} Holds(f, z_2) \equiv \;& Holds(f, z^+) \vee \\ & [Holds(f, z_1) \wedge \neg Holds(f, z^-)] \end{aligned}$$

So-called state update axioms define the effects of doing an action a in situation s by specifying the difference between the state of the successor situation $Do(a,s)$ and the state of the preceding situation s and allow to infer from an incomplete specification what follows of a resulting situation.

2.2 State probability

In (Thielscher, 2000), the basic fluent calculus has been extended by an explicit representation of both the knowledge of an agent and the effect of knowledge-producing actions. A different extension has been developed in (Jin and Thielscher, 2004) for representing beliefs of an agent. In this paper, we generalize these two approaches by a representation of the probability (any real-valued number from the interval $[0,1]$) of a state in a situation. For this we introduce the function

$$PState : \text{SIT} \times \text{STATE} \mapsto [0,1]$$

A foundational domain constraint stipulates that the state probabilities add up to 1 in every situation:[2]

$$\sum_z PState(s,z) = 1$$

As an example, consider a domain with the only fluents $Fragile(x)$ and $Broken(x)$ where $x \in \{Vase, Box\}$ and fluents $Fragile(x)$ and $Broken(x)$ shall denote the property of an object to be fragile and broken, respectively, along with the following axiomatization of the probability distribution at the initial situation S_0:

$$\begin{aligned} & (\exists x)\, Holds(Broken(x), z) \supset PState(S_0, z) = 0 \\ & Holds(Fragile(Vase), z) \wedge \neg(\exists x)\, Holds(Broken(x), z) \supset \\ & \qquad PState(S_0, z) = 0.4 \end{aligned} \tag{1}$$

Based on the representation of a probability distribution for a situation s, we can define the likelihood of a fluent f to hold in s as follows:[3]

[1] Throughout the paper, free variables are assumed to be universally quantified.

[2] For the sake of simplicity, we assume discrete probability distributions, which allows to sum over possible states; otherwise, an integral must be used.

[3] For the possibility of an infinite number of possible states the summations can be defined using a second-order formula. For the definition, a fluent calculus axiomatization with axiom schemata is used in order to have only countable many infinite states.

$$Bel(f,s) \stackrel{\text{def}}{=} \sum_{\{z|Holds(f,z)\}} PState(s,z)$$

For example, axioms (1) imply $Bel(Broken(Vase),S_0) = 0$ and $Bel(Fragile(Vase),S_0) = 0.8$.[4] The exact degree of belief in $Fragile(Box)$ is unknown, but it follows from the axiomatization that $0.4 \leq Bel(Fragile(Box),S_0) \leq 0.6$.[5] Macro $Bel(\phi,s)$ can be straightforwardly extended to the definition of the likelihood of more complex fluent formulas ϕ to hold in a situation s.

2.3 Probabilistic state update axioms

The solution to the frame problem can be extended to the fluent calculus with probabilities if the Markov assumption holds, that is, the (possibly noisy) effects of an action are independent of previous actions. The general probabilistic state update axiom is as follows:

$$PState(Do(a,s),z') = \sum_z PState(s,z) \cdot P(z,a,z')$$

where $P(z,a,z')$ is the probability that executing a in state z results in state z'. The state transition probability does not need to be specified for every single state, instead a succinct, factorized representation is sufficient. In order to facilitate a mapping to FLUX later on, we introduce the function $Case(s,z')$ in the specification of probabilistic effects. Each different "case" of an action $A(\vec{x})$ executed in situation s will be assigned a different number. A general effect specification for a noisy action $A(\vec{x})$ is of the form,

$$\begin{array}{ll}
(\exists \vec{y}_1) & \\
(\Phi_1(z) \supset & (z' = z - z_1^- + z_1^+ \supset \\
& P(z,a,z') = p_1 \wedge Case(s,z') = 1) \wedge \ldots \wedge \\
& (z' = z - z_k^- + z_k^+ \supset \\
& P(z,a,z') = p_k \wedge Case(s,z') = k)) \\
\wedge \ldots \wedge & \\
(\exists \vec{y}_n, j) & (\Phi_n(z) \supset \ldots
\end{array}$$

where $\Phi_i(z)$ is a state formula in z with free variables among $z,\vec{x},\vec{y}_i$; and $p_1,\ldots,p_k$ are probability values. We require that the conditions $\Phi_i(z)$ are exhaustive and mutually exclusive for each action $A(\vec{x})$. From our foundational axiom it follows additionally that within each condition the probabilities p_i sum to 1. As an example, consider the following effect specification for a noisy variant of a *Drop* action:

$$\begin{array}{l}
[Holds(Fragile(x),z) \supset \\
\quad (z' = z - Fragile(x) + Broken(x) \supset \\
\qquad P(z,Drop(x),z') = 0.9 \wedge Case(s,z') = 1) \\
\quad \wedge (z' = z \supset P(z',Drop(x),z') = 0.1 \wedge Case(s,z') = 2)] \\
\wedge \\
[\neg Holds(Fragile(x),z) \supset \\
\quad (z' = z \supset P(z',Drop(x),z') = 1.0 \wedge Case(s,z') = 3)]
\end{array}$$

Put in words, if x is fragile, then it will break with a probability of 0.9, otherwise the state, or more precisely the probability distribution, remains the same. Applied to the axiomatization of the initial probability distribution specified in (1), the probabilistic state update axioms entails, with $S_1 = Do(Drop(Vase),S_0)$,[6]

$$\begin{array}{l}
\neg Holds(Fragile(Vase),z') \wedge \\
[Holds(Broken(x),z') \equiv x = Vase] \\
\quad \supset PState(S_1,z') = 0.36
\end{array}$$

Since all other states containing $Broken(Vase)$ have probability 0, and because there are two states satisfying the condition of the implication above, it follows that $Bel(Broken(Vase),S_1) = 0.72$. Due to space limitations, we omit the other probabilities.

The effect of (possibly noisy) sensing actions is represented in the probabilistic fluent calculus as standard Bayesian update (the details have to be omitted).

3 Inferring Probabilistic Knowledge in FLUX

Our logical specification presented above exhibits nice properties, like e.g., the factorized representation. We now describe our computational approach, where we keep these advantages and are able to infer the result of executing action sequences always using only a single and compact probabilistic knowledge state.

3.1 The basic fluent calculus executor

The fluent calculus executor (FLUX) is an agent programming language which is formally grounded in the theory of the fluent calculus and has been formulated as Constraint Logic Program (for details see (Thielscher, 2005a)). The incomplete knowledge that an agent has of the state of its environment, is encoded

[4] To see why, note that with the additional fluent $Fragile(Box)$ there are four states with probability greater zero and exactly two of them satisfy the condition of the second implication in (1). Our approach would also allow for the direct specification of $Bel(Fragile(Vase),S_0) = 0.8$ instead of the second implication in (1). Together with the first implication in (1) this specification would imply the same conclusions regarding the belief of fluent *Fragile*.

[5] This can be seen from the fact that there is a state with probability 0.4 in which $Fragile(Box)$ does not hold, and also a state with probability 0.4 in which $Fragile(Box)$ does hold.

[6] To verify the resulting probability values, the reader may calculate the updated probability for each of the four possible states in situation S_0 in which no instance of $Broken(x)$ holds.

in FLUX by open lists (i.e., lists with a variable tail) of fluents. These lists are accompanied by constraints for negated or disjunctive state knowledge, as well as for variable range restrictions.

Just like in the fluent calculus, the effects of actions are encoded as state update axioms. For this purpose, the auxiliary predicate `update(Z1,P,N,Z2)` has been defined in FLUX, whose semantics is given by the fluent calculus update equation `Z2` $= ($`Z1` $-$ `N`$) +$ `P`. On this basis, the agent programmer can easily implement the individual, domain-dependent update axioms by clauses which define the predicate `state_update(Z1,A,Z2,S)`, where the last argument `S` denotes the returned sensing value. This parameter can be empty for non-sensing actions.

The foundational predicates for knowledge in FLUX have been carefully designed in such a way that every condition can be immediately evaluated in the current state while allowing for efficient constraint solving without the need to represent every possible state in FLUX. Instead, one knowledge state suffices (Thielscher, 2005a).

3.2 Adding probability to FLUX

Existing FLUX constraints are not sufficient to define arbitrary probabilistic dependencies among fluents. In order to encode both the probability of state properties to hold as in Section 2.2 and probabilistic state updates as in Section 2.3, probabilistic (causal) relations between fluents must be contained in the encoding of the knowledge of an agent. To achieve this while retaining the computational advantages of knowledge states in FLUX, we associate a Bayesian network, which has possibly disconnected subgraphs, to such a state and define a FLUX probabilistic state as:

Definition 1 A FLUX probabilistic state pz is a list

$$[f_1 : p_1, \ldots, f_n : p_n | z1]$$

of pairwise different fluents $(n \geq 0)$, each with its corresponding probability, along with a Bayesian network associated to $z1$. This network represent conditional probabilities between the fluents. □

Bayesian networks are an efficient way to represent (conditional) independence between variables (Pearl, 1988). Since we only have to encode the (relatively few, in a modular domain) local dependencies between fluents in a FLUX probabilistic state, we can avoid the need for an explicit representation of the global state space. To denote the conditional probability tables (CPTs) in our network, we actually use decision trees in our implementation. Decision trees can often avoid the local exponential representation of CPTs. For the moment, we make the following restrictions for the encoding in FLUX, which we would like to lift in the future:[7]

1. We are only using ground fluents in the networks. To arrive at ground fluents from a given domain description, we ground out all variables.[8]
2. We assume only finitely many possible states. For continuous values we would employ discretization.

Given the above restrictions and a situation, we can give a mapping τ from the full joint probability distribution over possible states in the fluent calculus to a FLUX probabilistic state: Given a situation s, the function $\pi(z) : \text{STATE} \rightarrow [0,1]$ with $(\forall z)(PState(s,z) = \pi(z))$ defines an full joint probability over all possible states z in s. The probability is defined over the literals contained in the possible state: $\pi(z) = P(l_1 \wedge \ldots \wedge l_n)$ with $l_i = f_i$ if $Holds(f_i, z)$ and $l_i = \neg f_i$ if $\neg Holds(f_i, z)$. By the product rule, the function π can also be written as: $\pi(z) = P(l_1 | l_2 \wedge \ldots \wedge l_n) \cdot \ldots P(l_i | l_{i+1} \wedge \ldots \wedge l_n) \ldots \cdot P(l_n)$. From a given function $\pi(z)$ we can induce a FLUX probabilistic state by letting,[9]

$$\tau(\pi(z)) = [f_1 : p_1, \ldots, f_n : p_n | z1]$$

where every $p_i = \sum_{\{z | Holds(f_i, z)\}} \pi(z)$, the conditional probabilities $P(l_1 | l_2 \wedge \ldots \wedge l_n), \ldots, P(l_i | l_{i+1} \wedge \ldots \wedge l_n), \ldots$ are represented by the network associated to $z1$, and the marginal probability of literal $P(l_i) = P(p_i)$ if l_i is true and otherwise $P(l_i) = 1 - P(p_i)$. The literals in $\pi(z)$ are ordered in an alphabetic order before the mapping to avoid cycles in the network.

To answer queries to a Bayesian network, we use a variant of the well-known variable elimination (VE) algorithm, the contextual VE algorithm (Poole and Zhang, 2003). This algorithm works with so-called confactors which can be seen as branches of a decision tree. Due to lack of space, we have to omit the details here.

3.3 Probabilistic state update in FLUX

As already noted in (Boutillier et al., 1999), a Bayesian net representation is equivalent in expressive power to a general stationary transition matrix

[7] For the sake of easier exposition in this paper, we assume complete information about the probabilities of the possible states. We can handle incompletely specified probabilistic state knowledge using intervals.

[8] In the fluent calculus axiomatization, we use an appropriate domain closure axiom and restrict the number of objects to finitely many.

[9] It is also possible to give a mapping τ^{-1}.

model. We use so-called two-stage Bayesian networks (2TBN) for the representation of our actions in FLUX, where there are pre-action variables and corresponding post-action variables. Directed arcs between those types of variables indicate probabilistic dependencies. Every 2TBN has a natural interpretation as a stationary Markov chain, where the conditional distributions in the network are state-transition probabilities and the marginal distributions are initial state distributions. Furthermore, if a network representing an action contains a case node to distinguish the individual conditional outcomes of an action as defined in Section 2.3, it suffices to have a 2TBN without any arcs between the post-action variables (Boutillier et al., 1999).

State updates for non-sensing[10] actions in FLUX require updates of the 2TBN. The probabilistic effect specifications in the fluent calculus (see Section 2.3) contain all necessary information to construct in a general way an equivalent 2TBN representation:

1. The probabilities of all the cases have to be inferred and represented in a case node in the network. These probabilities depend only on the conditions Φ_i and can easily be represented by a decision tree. The value is given by the state transition probabilities $P(z,a,z')$.
2. The probability of every post-action variable now only depend on the case and (possibly) on whether its corresponding pre-action variable was true or false. Once the case is known, this deterministic effect can also be represented by a decision tree.

As an example, recall the noisy variant of the $Drop(x)$ action defined in Section 2.3, which, if instantiated by $\{x/Vase\}$ in situation S_0, gives rise to the net depicted in Figure 1. The initial probabilities for the fluents $Broken(Vase)$ and $Fragile(Vase)$ are given as in the example initial situation S_0. For this example situation, the induced initial FLUX probabilistic state does not have to contain any initial Bayesian network as an independence between the fluents is assumed. In general, there may be a network with initial conditional probabilities between the fluents.

Since we do forward reasoning for probabilistic projection and for planning, the dependencies of the fluents in the old state (pre-action) can be ignored after the execution of an action. Only the causal relations among the fluents in the new state (post-action) need to be kept. In this way, the associated network is kept small for the sake of efficient inferencing. To

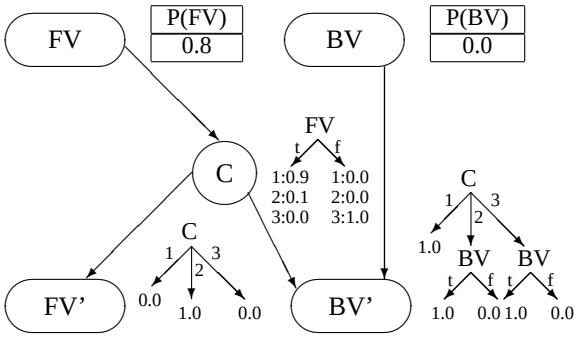

Figure 1: The nodes FV, BV and C stand, respectively, for $Fragile(Vase)$, $Broken(Vase)$ and $Case$. The primed nodes are the post-action variables.

obtain the reduced network, we have to compute the conditional probabilities between the fluents in the new state. This can be achieved using our implemented contextual variable elimination algorithm and appropriate fluent nodes of the new state as evidence variables. The computation is possible as long as cycles are avoided in the construction of the new, reduced network. The resulting network contains only fluents of the new state. Furthermore, we can infer all marginal probabilities for the nodes in the new network. As the reduced network and the updated belief of the fluents represent the same information about the new state, we can now dispose of the old, initial network. The resulting FLUX probabilistic state including the updated network now determines a function $PState(Do(a,s),z)$ which corresponds exactly to the result of applying the general probabilistic state update axiom of the fluent calculus as defined in Section 2.3. As we use the contextual VE algorithm on a network with possibly disconnected subgraphs, our algorithm can have exponential time and space complexity in the size of the network in the worst case. Here, the size of the network is defined as the number of decision trees. In practice, as we always reduce the network after each action and each of our agent's actions only affect few fluents in our application domains, our algorithm can compute the necessary inferences efficiently.

We can now answer the example query from Section 2.3 in FLUX (here presented in standard Prolog notation):[11]

```
?- state_update(Z0,drop(vase),Z1,[]).
      Belief in Z1 of fragile(vase) is 0.08
      Belief in Z1 of broken(vase) is 0.72
```

The computed answer shows the expected likelihood

[10] Sensing actions do not change the network structure, they only update the likelihood of fluents according to standard Baysian update.

[11] For the sake of simplicity in this paper, we show only the inferences for the object $Vase$ here and assume a degree of belief for $Fragile(Box)$ of 0.5.

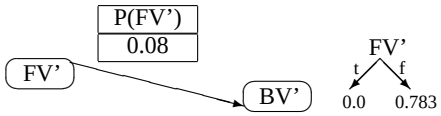

Figure 2: Node $Fragile(Vase)$ has causal influence on fluent $Broken(Vase)$.

of the fluents and associates the new, reduced network depicted in Figure 2 to state `Z1` and disposes the network in Figure 1. This answer represents the same function $PState(S_1, z)$ as in the fluent calculus with $S_1 = Do(Drop(Vase), S_0)$.

Suppose, for example, that now the action $Drop(Vase)$ were executed a second time, leading to the situation $S_2 = Do(Drop(Vase, Do(Drop(Vase), S_0)))$, then we would use the network in Figure 2 to infer the updated probabilities for $State(S_2)$. In this way, the network can be reduced after any further application of this action, so that we always obtain a network containing no more than two nodes.

Soundness of the FLUX encoding

As a consequence of the mappings described in Sections 3.2 and 3.3, we have the following correctness result.

Proposition 2 *For an arbitrary situation s, let $(\forall z)(PState(s,z) = \pi_1(z))$ and the induced FLUX probabilistic state be $\tau(\pi_1(z))$. Let $[\alpha_1, \ldots, \alpha_n]$ be a sequence of ground actions where for every action α_i we have a corresponding 2TBN representation obtained as described above. Let the FLUX probabilistic state $\tau(\pi(z))$ be the result of computing with FLUX the state updates of α_1 until α_n starting in the initial state $\tau(\pi_1(z))$. Then from the fluent calculus axiomatization it follows:*

$$(\forall z)(PState(s,z) = \pi_1(z)) \supset \\ (\forall z)(PState(Do([\alpha_1, \ldots, \alpha_n], s), z) = \pi_2(z))$$

Proof Sketch: The proof is by induction on n.

If $n = 0$, then $\tau(\pi_1(z)) = \tau(\pi_2(z))$. As no action is executed, from the fluent calculus axiomatization it follows that $\pi_1(z) = \pi_2(z)$.

Suppose the claim holds for $n-1$. We are given a FLUX probabilistic state and have to infer the effects of action a_n. It is easy to verify that we can transform both the probabilistic state and the action a_n in a multiset of confactors. The correct contextual variable elimination algorithm of (Poole and Zhang, 2003) can be employed to infer the answer to arbitrary queries about probabilistic dependencies between the random variables.

We restrict our attention to the post-action variables in the multiset of confactors and apply the contextual variable elimination algorithm to compute their marginal probabilities and the conditional probabilities between them. The computed conditional probabilities are again represented in FLUX by confactors. Only the result of these computations is integrated into the new FLUX probabilistic state $\tau(\pi_2(z))$.

The state update of action a_n was correctly computed in FLUX and together with the induction hypothesis this implies the claim. ■

4 Conclusion

We have presented a formalism and a logic programming approach for agents to represent and reason with probabilistic knowledge. Our computational approach combines knowledge states with Bayesian networks and allows to do all inferences with a single such state rather than an explicit encoding of the entire space of all possible states. For each projected action execution, we only have to update the degree of belief for the fluents involved in this action and those fluents connected to the former within the same subgraph of the network. As we have a small Bayesian network, this can be computed efficiently. Additionally, we update the network to keep it small. We can also express, and reason with, incompletely specified state probabilities.

Other logic programming approaches for agent programming, e.g., (Reiter, 2001b; Shanahan and Witkowski, 2000), lack an explicit notion of a state. Knowledge of the current state is represented indirectly via the initial conditions and the actions which the agent has performed up to now. As a consequence, the entire history of actions is needed when evaluating a probability of a property in an agent program. Moreover, the entire state space of all the resulting states has to be considered when inferring probabilities in (Reiter, 2001a; Grosskreutz and Lakemeyer, 2000; Baier and Pinto, 2003) or assessing plans in (Kushmerick et al., 1995), which easily leads to long computation times even for small examples.

Alternative approaches to combine reasoning about actions and probability include (Pearl, 2000; Tran and Baral, 2004; Gardiol and Kaelbling, 2004), but they either cannot deal with action sequences, or stay entirely within propositional logic, or cannot express uncertainty over the state probability distribution. Moreover, none of these approaches has been embedded in a general agent programming language like FLUX.

Decision theoretic regression with 2TBN and influence diagrams based on such networks (Boutillier et al., 1999) are also concerned with expressing temporal probabilistic dependencies. However, there are

some important differences to our approach. Except for a preliminary treatment in (Boutilier and Poole, 1996), all the literature concerns only fully observable MDPs while our approach represents POMDPs. There is no notion of a precondition of an action in (Boutillier et al., 1999). In our approach, we can define states for which certain actions in a future planning extension should not even be considered for selection. While we inherit the representational and inferential solution to the frame problem from the solution in the standard fluent calculus, using only 2TBN as in (Boutillier et al., 1999) one must explicitly assert that fluents unaffected by a specific action persist in value, although the representational frame problem (but not the inferential one) can be solved by automated assertions (Boutilier and Goldszmidt, 1996).

Our approach should be extended in future work to allow for planning under uncertainty. To construct plans which achieve a specific condition with a probability above a threshold, we could apply conditional planning as defined in (Thielscher, 2005b) or use iterative planning with loops in the sense of (Levesque, 2005). It would also be possible to give plan skeletons in FLUX similar to (Grosskreutz and Lakemeyer, 2000), which can drastically reduce planning time. The verification that a plan satisfies a goal with some probability threshold can then be inferred efficiently with our approach.

For additional future work, we intend to investigate to which extent we can avoid grounding a first-order knowledge state as much as possible and use a first oder algorithm to query our Bayesian networks (de Salvo Braz et al., 2007).

REFERENCES

Bacchus, F., Halpern, J., and Levesque, H. (1999). Reasoning about noisy sensors and effectors in the situation calculus. *Artificial Intelligence*, 111(1–2):171–208.

Baier, J. A. and Pinto, J. (2003). Planning under uncertainty as Golog programs. *J. Exp. Theor. Artif. Intell.*, 15(4):383–405.

Boutilier, C. and Goldszmidt, M. (1996). The frame problem and Bayesian network action representations. In *Proceedings of the Canadian Conference on Artificial Intelligence (CSCSI)*.

Boutilier, C. and Poole, D. (1996). Computing optimal policies for partially observable decision processes using compact representations. In *Proceedings of the 13-th National Conference on Artificial Intelligence (AAAI)*, pages 1168–1175, Portland, Oregon, USA.

Boutillier, C., Dean, T., and Hanks, S. (1999). Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. *Journal of Artificial Intelligence Research*, 11:1–94.

de Salvo Braz, R., Amir, E., and Roth, D. (2007). Lifted first-order probabilistic inference. In Getoor, L. and Taskar, B., editors, *Introduction to Statistical Relational Learning*. MIT Press.

Gardiol, N. H. and Kaelbling, L. P. (2004). Envelope-based planning in relational MDPs. In *Advances in Neural Information Processing Systems 16 (NIPS-03)*, Vancouver, CA.

Grosskreutz, H. and Lakemeyer, G. (2000). Turning high-level plans into robot programs in uncertain domains. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*.

Jin, Y. and Thielscher, M. (2004). Representing beliefs in the fluent calculus. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 823–827, Valencia, Spain. IOS Press.

Kushmerick, N., Hanks, S., and Weld, D. S. (1995). An algorithm for probabilistic planning. *Artificial Intelligence*, 76(1-2):239–286.

Levesque, H. (2005). Planning with loops. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, Edinburgh, Scotland.

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

Pearl, J. (2000). *Causality: Models, Reasoning, and Inference*. Cambridge University Press.

Poole, D. and Zhang, N. L. (2003). Exploiting contextual independence in probabilistic inference. *Journal of Artificial Intelligence Research*, 18:263–313.

Reiter, R. (2001a). *Knowledge in Action*. MIT Press.

Reiter, R. (2001b). On knowledge-based programming with sensing in the situation calculus. *ACM Transactions on Computational Logic*, 2(4):433–457.

Shanahan, M. and Witkowski, M. (2000). High-level robot control through logic. In *Proceedings of the International Workshop on Agent Theories Architectures and Languages (ATAL)*, volume 1986 of *LNCS*, pages 104–121, Boston, MA. Springer.

Thielscher, M. (1999). From situation calculus to fluent calculus: State update axioms as a solution to the inferential frame problem. *Artificial Intelligence*, 111(1–2):277–299.

Thielscher, M. (2000). Representing the knowledge of a robot. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 109–120, Breckenridge, CO. Morgan Kaufmann.

Thielscher, M. (2005a). FLUX: A logic programming method for reasoning agents. *Theory and Practice of Logic Programming*, 5(4–5):533–565.

Thielscher, M. (2005b). *Reasoning Robots: The Art and Science of Programming Robotic Agents*, volume 33 of *Applied Logic Series*. Kluwer.

Tran, N. and Baral, C. (2004). Encoding probabilistic causal model in probabilistic action language. In *Proceedings of the 19-th National Conference on Artificial Intelligence (AAAI)*, pages 305–310.