Extending DTGolog to Deal with POMDPs

Gavin Rens[1,2], Alexander Ferrein[3], Etienne van der Poel[1]

[1] School of Computing, Unisa, Pretoria, South Africa
[2] Knowledge Systems Group, Meraka Institute, CSIR, Pretoria, South Africa
[3] Knowledge-Based Systems Group, RWTH Aachen University, Aachen, Germany
grens@csir.co.za
ferrein@cs.rwth-aachen.de
evdpoel@unisa.ac.za

Abstract

For sophisticated robots, it may be best to accept and reason with noisy sensor data, instead of assuming complete observation and then dealing with the effects of making the assumption. We shall model uncertainties with a formalism called the *partially observable Markov decision process* (POMDP). The planner developed in this paper will be implemented in *Golog*; a theoretically and practically 'proven' agent programming language. There exists a working implementation of our POMDP-planner.

1. Introduction

If a robot or agent can perceive every necessary detail of its environment, its model is said to be *fully observable*. In many practical applications this assumption is good enough for the agent to fulfill its tasks; it is nevertheless unrealistic. A more accurate model is a *partially observable model*. The agent takes into account that its sensors are imperfect, and that it does not know every detail of the world. That is, the agent can incorporate the probabilities of errors associated with its sensors, and other uncertainties inherent in perception in the real world, for example, obscured objects. If an agent or robot cannot represent the uncertainties inherent in perception, it has to *assume* perfect perception. This assumption either might lead to spurious conclusions or the necessity for additional methods that keep the agent's reasoning reasonable. For sophisticated robots or agents, it may be best to accept and reason with noisy sensor data.

One model for reasoning under uncertainty with partial observability is the *partially observable Markov decision process* (POMDP). In this paper we present POMDP models based on the robot programming and planning language Golog [1]. In particular, we extend DTGolog [2], a Golog dialect. DTGolog employs a notion of perfect perception; we extent it with a notion of graded belief.

The rest of the paper is organised as follows. In the next section we briefly introduce the situation calculus and present the robot programming and planning language DTGolog, before we formally define POMDPs in Section 3. In Section 4 we present some related work. Section 5 introduces the predicate $BestDoPO$ which defines the semantics of the POMDP planner in Golog. Section 6 presents a simple example of how planning under partial observability is conducted. We conclude with Section 7.

2. The Situation Calculus and DTGolog

The situation calculus is a first order logic dialect for reasoning about dynamical systems based on agent actions. The outcomes of a bout of reasoning in the situation calculus are meant to have effects on the environment outside the agent. When an agent or robot performs an action, the truth value of certain predicates may change. Predicates whose value can change due to actions are called *fluents*. Fluents have the *situation term* s as the last argument.

A special function symbol do is defined in the situation calculus. $do(a, s)$ is the name of the situation (that the agent is in) given the agent does action a in situation s. Note that $do(a_2, do(a_1, s))$ is also a situation term, where a_2 and a_1 are actions.

To reason in the situation calculus, one needs to define an initial knowledge base (KB). The only situation term allowed in the initial KB is the special *initial situation* S_0. S_0 is the situation before any action has been done.

There are two more formulas that need our attention:

1. The *precondition axioms* are formulas of the form $Poss(a, s)$, which means action a is possible in situation s ($\neg Poss(a, s)$ means it is not possible). Precondition axioms need to be defined for each action.
2. *Successor-state axioms* are formulas that define how fluents' values change due to actions. There needs to be a successor-state axiom for each fluent, and each such successor-state axiom mentions only the actions that have an effect on the particular fluent.

Please refer to [3] for a detailed explication of the situation calculus, including a description of the famous *frame problem* and how the *basic action theory* is a solution to this problem. Alternatively, refer to [4] for a one-chapter coverage of the situation calculus.

Decision-theoretic Golog (DTGolog) [2] is an extension to Golog to reason with probabilistic models of uncertain actions. The formal underlying model is that of fully observable Markov decision processes (MDPs).

Golog is an agent programming language (APL) developed by [1]. It is based on the situation calculus. It has most of the constructs of regular procedural programming languages (iteration, conditionals, etc.). What makes it different from other programming languages is that it is used to specify and control *actions* that are intended to be executed in the real world or a simulation of the real world. That is, Golog's main variable type is the action (not the number).

Complex actions can be specified by combining atomic actions. The following are all complex actions (where a subscripted is an atomic action and φ is a sentence):

- while φ do a_1 (iteration of actions);
- $\varphi?:a_1$ (test action);
- if φ then a_1 else a_2 (conditional actions);
- $a_1; a_2; \ldots; a_k$ (sequence of actions);
- $a_1 | a_2$ (nondeterministic choice of actions);
- $\wp x.(a_1)$ (nondeterministic finite choice of arguments—of x in a_1);

$Do(A, s, s')$ holds if and only if the complex action A can terminate legally in s' when started in situation s.

The DTGolog algorithm is defined with $BestDo$ predicates, taking on the role of Golog's Do. The DTGolog interpreter however, does not simply 'perform' the program (complex action) given it, but calculates an optimal policy based on an optimization theory: the forward search value iteration algorithm for fully observable MDPs. [1] capture the nondeterministic aspect of MDPs with predicates $stochastic$, and $prob$. $prob(n, p, s)$ determines the probability p with which action n is the outcome in some situation s. (In this section we define prob as a *function* that *returns* the probability.) Let $choice'(a) = \{n_1, \ldots, n_k\}$ (derived from $stochastic$) be the k actions that nature could 'choose' (the actual action performed) for the agent's *intended* action a. For stochastic action a:

$$BestDo(a; r, s, h, \pi, v, pr) \doteq$$
$$\exists \pi', v'.BestDoAux(choice'(a), a, rest, s, h, \pi', v', pr) \wedge$$
$$\pi = a; senseEffect(a), \pi' \wedge v = reward(s) + v'.$$

$a; r$ is the input program, with a the first action in the program and r the rest of the program; s is the situation term; the agent designer needs to set the number of steps (actions) h for which a policy is sought—the *planning horizon*; π returns the policy; v is the expected reward for executing π; pr returns the probability with which the input program will be executed as specified, given the policy and given the effects of the environment. $senseEffect(a)$ is a pseudo-action included in the formalism to ensure that the formalism stays in the fully observable MDP model. $BestDoAux$ deals with each of the possible realizations of a stochastic action:

$$BestDoAux(\{n_1, \ldots, n_k\}, a, r, s, h, \pi, v, pr) \doteq$$
$$\neg Poss(n_1, s) \wedge BestDoAux(\{n_2, \ldots, n_k\}, a, r, s, h, \pi, v, pr) \vee$$
$$Poss(n_1, s) \wedge$$
$$\exists \pi', v', pr'.BestDoAux(\{n_2, \ldots, n_k\}, a, r, s, h, \pi', v', pr') \wedge$$
$$\exists \pi_1, v_1, pr_1.BestDo(r, do(n_1, s), h-1, \pi_1, v_1, pr_1) \wedge$$
$$senseCond(n_1, \varphi_1) \wedge \pi = \textbf{if } \varphi_1 \textbf{ then } \pi_1 \textbf{ else } \pi' \textbf{ endif} \wedge$$
$$v = v' + v_1 \cdot prob(n_1, a, s) \wedge pr = pr' + pr_1 \cdot prob(n_1, a, s).$$

For any action n, $senseCond(n, \varphi)$ supplies a sentence φ that is placed in the policy being generated. φ holds if and only if the value returned by the sensor can verify that action n was performed.

When either of two actions δ_1 and δ_2 can be performed, the policy associated with the action that produces the greater value (current sum of rewards) is preferred and that action is included in the determination of the final policy π. This formula captures the idea that is at the heart of the *expected value maximization* of decision theory:

$$BestDo([\delta_1|\delta_2]; r, s, h, \pi, v, pr) \doteq$$
$$\exists \pi_1, v_1, pr_1.BestDo(\delta_1; r, s, h, \pi_1, v_1, pr_1) \wedge$$
$$\exists \pi_2, v_2, pr_2.BestDo(\delta_2; r, s, h, \pi_2, v_2, pr_2) \wedge$$
$$((v_1, \delta_1) \geq (v_2, \delta_2) \wedge \pi = \pi_1 \wedge v = v_1 \wedge pr = pr_1) \vee$$
$$((v_1, \delta_1) < (v_2, \delta_2) \wedge \pi = \pi_2 \wedge v = v_2 \wedge pr = pr_2)).$$

3. POMDP defined

3.1. The model

In partially observable Markov decision processes (POMDPs) actions have nondeterministic results and observations are uncertain. In other words, the effect of some chosen action is somewhat unpredictable, yet may be predicted with a probability of occurrence. And the world is not directly observable; some data are observable, and the agent infers how likely it is that the state of the world is in some specific state. The agent thus believes to some degree—for each possible state—that it is in that state, but it is never certain exactly which state it is in. Furthermore, a POMDP is a *decision* process and thus facilitates making decisions as to which actions to take, given its previous observations and actions.

Formally, a POMDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O}, b_0 \rangle$ with the following seven components (see e.g., [5, 6]): (1) $\mathcal{S} = \{s_0, s_1, \ldots, s_n\}$ is a finite set of states of the world; the state at time t is denoted s^t; (2) $\mathcal{A} = \{a_1, a_2, \ldots, a_k\}$ is a finite set of actions; (3) $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\mathcal{S})$ is the *state-transition function*, giving for each world state and agent action, a probability distribution over world states; (4) $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the *reward function*, giving the immediate reward that the agent can gain for any world state and agent action; (5) $\Omega = \{o_0, o_1, \ldots, o_m\}$ is a finite set of observations the agent can experience of its world; (6) $\mathcal{O} : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\Omega)$ is the *observation function*, giving for each agent action and the resulting world state, a probability distribution over observations; and (7) b_0 is the initial probability distribution over all world states in $\mathcal{S}$.

An important function is the function that updates the agent's belief: [5] call this function the *state estimation* function $SE(b, a, o)$. b is a set of pairs (s, p) where each state s is associated with a probability p, that is, b is a probability distribution over the set S of all states. b can be called a *belief state*. SE is defined as

$$b^t(s') = \frac{\mathcal{O}(s', a, o) \sum_{s \in S} T(s, a, s') b^{t-1}(s)}{Pr(o|a, b)}, \qquad (1)$$

where $b^t(s')$ is the probability of the agent being in state s' at time-step t. (Action and observation subscripts have been ignored.) Equation (1) is derived from the Bayes Rule. $Pr(o|a, b)$ in the denominator is a normalizer; it is constant with time. SE returns a new belief distribution for every action-observation pair. SE captures the Markov assumption: a new state of belief depends only on the immediately previous observation, action and state of belief.

3.2. Determining a policy

For any set of sequences of actions, the sequence of actions that results in the highest expected reward is preferred. The *optimality prescription* of utility theory states: Maximize "the expected sum of rewards that [an agent] gets on the next k steps," [5]. That is, an agent should maximize $E\left[\sum_{t=0}^{k-1} r_t\right]$ where r_t is the reward received on time-step t.

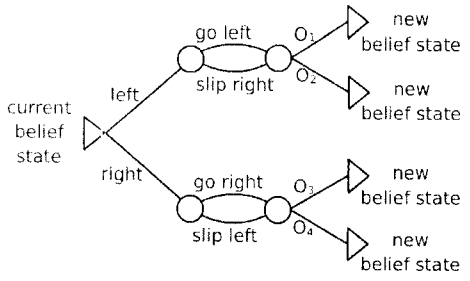

Figure 1: One tier of a POMDP-decision-tree.

When the states an agent can be in are belief states, we need a reward function over belief states. We derive $Rb(a, b)$ from the reward function over world states, such that a reward is proportional to the probability of being in a world state:

$$Rb(a, b) = \sum_{s \in S} R(a, s) \times b(s). \quad (2)$$

Now the aim of using POMDP models is to determine recommendations of 'good' actions or decisions. Such recommendations are called a *policy*. Formally, a policy (π) is a function from a set B of all belief states the agent can be in, to a set of actions: $\pi : B \rightarrow A$. That is, actions are *conditioned* on beliefs. So given b_0, the first action a' is recommended by π. But what is the next belief state? This depends on the next observation. Therefore, for each observation associated with a', we need to consider a different belief state. Hence, the next action, a'', actually depends on the observations associated with (immediately after) a'. In this sense, a policy can be represented as a *policy tree*, with nodes being actions and branches being observations. The above function is thus transformed to $\pi : \mathcal{O} \rightarrow A$. Now once we *have* a policy, it is independent of the agent's beliefs, except its initial belief.

Let $V_{\pi,t}(s)$—the *value function*—be the expected sum of rewards gained from starting in world state s and executing policy π for t steps. If we define a value function over *belief* states as $Vb_{\pi,t}(b) = \sum_{s \in S} V_{\pi,t}(s) \times b(s)$, we can define the *optimal* policy $\pi*$ with planning horizon h (set $t = h$) as

$$\pi* = argmax_{\pi}(Vb_{\pi,h}(b_0)) \quad (3)$$

(from the initial belief state)—the policy that will advise the agent to perform actions (given any defined observation) such that the agent gains maximum rewards (after h actions).

To implement Equation (3), the authors make use of a decision tree (there are other methods). DTGolog uses a similar approach: *forward search value iteration*. An example sub-decision-tree (one tier) is shown in Figure 1. This example is based on an environment and agent model where the agent can only go left or right and each of its two actions has two possible realizations in the environment; also, the agent may make two kinds of observations (O_1 and O_2) if it chose to go left, and another two kinds of observations (O_3 and O_4) if it chose to go right.

Belief states (triangles) in the decision tree are decision nodes, that is, at these nodes, the agent can choose an action (make a decision). Circles are chance nodes, that is, certain events occur, each with a probability (chance) such that any one event at one chance node will definely happen (probabilities of branches leaving a chance node, sum to 1).

In Decision Analysis (see e.g., [7]), we roll back a decision tree to 'decide' the action. In any decision tree, for each action-observation pair, there is a tier of sub-decision-trees. That is, when considering N actions in a row, a decision tree with N tiers would be required. There is a unique path from the initial decision node to each leaf node, and at each belief state encountered on a path, a reward is added, until (and including) the leaf belief state. At this point, the agent knows the total reward the agent would get for reaching that final state of belief. Each of the belief states is reachable with some probability.

At each decision node, a choice is committed to. We iteratively roll back—from last decision nodes to first decision node. The agent can in this way decide at the first decision node, what action to take. Each subtree rooted at the end of the branches representing the agent's potential action, has an associated expected reward. The action rooted at the subtree with the highest expected reward, should be chosen.

As the decision tree is rolled back, the best decision/action is placed into the policy, conditioned on the most recent possible observations. Using such a *policy tree* (generated from a decision tree), the agent can always choose the appropriate action given its last observation. This is the essence of the theory on which our POMDP planner is based.

4. Related work

In the following, we present some related work dealing with reasoning under uncertainty. As there exists a large body of work in this field, we concentrate in particular on approaches for reasoning under uncertainty in the situation calculus and Golog.

[8]'s idea of representing beliefs is simple yet important. Intuitively, their aim is to represent an agent's uncertainty by having a notion of which configuration of situations are currently possible; the possible worlds framework. Then further, each possible world is given a likelihood weight. With these notions in place, they show how an agent can have a belief (a probability) about any sentence in any defined situation. Their work does not, however, cover planning.

Reiter [3] describes how to implement MDPs as well as POMDPs in the situation calculus. He defines the language *stGolog*, which stands for 'stochastic Golog'. Nevertheless, Reiter does not provide a method to automatically generate (optimal) policies, given a domain and optimization theory; he only provides the tools for the designer to program by hand policies for partially observable decision domains.

Grosskreutz shows how the Golog framework "can be extended to allow the projection of high-level plans interacting with noisy low-level processes, based on a probabilistic characterization of the robot's beliefs," [9]. He calls his extension to Golog *pGolog*. The belief update of a robot's epistemic state is also covered by [9]. (PO)MDPs are not employed in pGolog. Instead, he does probabilistic projection of specific programs. He does however make use of expected utility to decide between which of two or three or so programs to execute (after simulated scenarios).

In [11], Ferrein and Lakemeyer present the agent programming language *ReadyLog*. Approximately ten years after Golog's birth, ReadyLog combines many of the disparate useful features of the various dialects of Golog into one package. ReadyLog has been implemented and successfully used in robotic soccer competitions and a prototype domestic robot.

Whereas DTGolog [2] models MDPs—a useful model in robotics, as most robots operate in environments where actions have uncertain outcomes—our new dialect models *belief-MDPs*. A belief-MDP is one perspective of POMDPs, where the states that are being reasoned over are *belief* states and not the *world* states of MDPs. More detail concerning the semantics of DTGolog is given in Section 2.

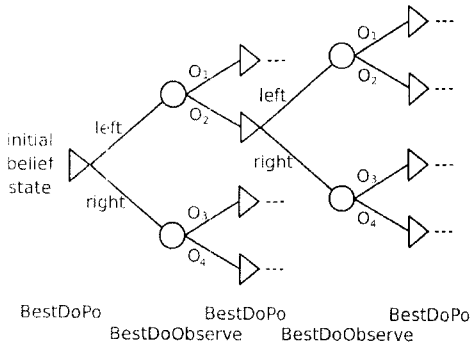

Figure 2: $BestDoPO$ represented as a POMDP-decision-tree.

Very related to our approach is the approach of [10]. Finzi and Lukasiewicz present a game-theoretic version of DTGolog to operate in partially observable domains. They call this extension *POGTGolog*. As far as we know, this is the only Golog dialect that can take partially observable problems as input, that is, that has some kind of POMDP solver for agent action planning. POGTGolog deals with multiple agents. Our work is different from theirs, as we concentrate on the single agent case and our agent is not restricted to game theory. For developers who prefer a Golog dialect for agent programming, but desire their robots or agents to operate with POMDP information, these developers cannot easily modify POGTGolog to work with single robots. Our work is not only a simplification of [10]; rather, we extend DTGolog, and use several elements in POGTGolog—either directly or for inspiration.

5. Semantics of POMDPs in Golog

In this section we describe our extension to the original forward search value iteration algorithm as proposed in [2]. In the following, we extend the approach of DTGolog in such a way that it can also deal with partially observable domains. In particular, instead of using $BestDo$, we introduce a predicate $BestDoPO$ to operate on a belief state rather than on a world state. $BestDoPO(p, b, h, \pi, v, pr)$ takes as arguments a Golog program p, a belief state b and a horizon h, which determines the solution depths of the algorithm. The policy π as well as its value v and the success probability pr are returned by the algorithm.

The relation of $BestDoPO$ to a POMDP-decision-tree can be seen in Figure 2. The stochastic outcomes of actions has been suppressed for ease of presentation.

An example of how $BestDoPO$ may be called initially—with a program that allows the agent to choose between three actions a_1, a_2, a_3 (without constraints), with b_0 the initial belief state and with the user or agent requiring advice for a sequence of seven actions—is $BestDoPO(\textbf{while } true \textbf{ do } [a_1 \mid a_2 \mid a_3], b_0, 7, \pi, v, pr)$.

5.1. Basic definitions and concepts

A belief state b contains the elements (s, p); each element/pair is a possible (situation calculus) situation s together with probability p (as in [10]).

We use the idea of [10] and assume that an action is possible in a belief state, when it it possible in the situation which is part of the belief state, that is, $PossAct(a, b)$ iff $PossAct(a, s)$ (we rename the traditional $Poss$ to $PossAct$). We add the predicate $PossObs(o, a, s)$ to the action theory, which specifies when an observation o is possible (perceivable) in situation s, and define $PossObs(o, a, b)$ iff $PossObs(o, a, s)$, which defines when the observation is possible in belief state b, given an action a. The reader should clearly distinguish between preconditions for observations, $PossObs(o, a, s)$ and for actions, $PossAct(a, s)$. It is important to note that the b' in $PossObs(o, a, b')$ is the belief state reached *after* action a was executed. That is, if a was executed in b and b' is the new state reached, then $PossObs(o, a, b')$ says whether it is possible to observe o *after* a has been executed.

Next, we define a function symbol called $probNat(n, a, s)$ that is similar in meaning to the state transition function T of a Markov process. Our definition 'returns' a probability. It applies to all of nature's choices n, where s is the state in which the agent performs action a. Similarly, we introduce the function $probObs(o, a, s)$; the probability that o will be observed in s after a was executed in the previous situation.

Finally, we define $belObs$, which is the probability that the agent will observe some specified observation given its current beliefs and the sensor it activated: $belObs(o, a, b) \doteq \sum_{(s',p')\in b} p' \cdot probObs(o, a, s')$.

In the next section we briefly sketch our solution algorithm which calculates optimal policies under partial observability.

5.2. The partially observable $BestDo$

This subsection presents the key formulas in the definition of $BestDoPO$.

Considering possible observations after an action, we branch on all possible observations, given the robot's intended action a. $choiceObs'(a)$ 'returns' the set of observations that the robot may perceive: $\{o|choiceObs(o, a, s)$ for all $s \in S\}$. The reward function R is defined by (Eq. 2).

Probabilistic observation

$$
\begin{aligned}
&BestDoPo(a; rest, b, h, \pi, v, pr) \doteq \\
&\quad \neg PossAct(a, b) \wedge \pi = Stop \wedge v = 0 \wedge pr = 0 \ \vee \\
&\quad PossAct(a, b) \wedge \\
&\quad \exists \pi', v'. BestDoObserve(choiceObs'(a), \\
&\qquad a, rest, b, h, \pi', v', pr) \ \wedge \\
&\quad \pi = a; \pi' \wedge v = R(b) + v'.
\end{aligned}
$$

After a certain action a and a certain observation o_k, the next belief state is reached. At the time when the auxiliary procedure $BestDoObserve$ is called, a specific action, the set of nature's choices for that action and a specific observation associated with the action are under consideration. These elements are sufficient and necessary to update the agent's current beliefs. Inside $BestDoObserve$, the belief state (given a certain action and observation history) is updated via a belief state transition function (similar in vein to the state estimation function of Section 3, and the successor-state axiom for likelihood weights as given in [8]).

Belief update function

$$
\begin{aligned}
&b_{new} = BU(o, a, b) \doteq \\
&\quad \textbf{for each } (s, p) \in b \\
&\qquad \exists n, s^+, p^+.(s^+, p^+) \in b_{temp} : s^+ = do(n, s) \wedge \\
&\qquad choiceNat(n, a, s) \wedge PossAct(n, s) \wedge \\
&\qquad p^+ = p \cdot probObs(o, a, s^+) \cdot probNat(n, a, s) \\
&\quad \textbf{end for each} \\
&\quad b_{new} = normalize(b_{temp}).
\end{aligned}
$$

Figure 3: Four-state world; four states in a row. Initially the agent believes it is in each state with probabilities [0.04|0.95|0.00|0.01] corresponding to state position.

A major difference between the POMDP model as defined in Section 3 and the POMDP model we define here for the situation calculus, is that here the belief state is not a probability distribution over a *fixed* set of states. If a situation (state) was part of the belief state to be updated, it is removed from the new belief state, and situations (states) that are 'accessible' from the removed situation via $choiceNat(n, a, s)$ *and* are executable via $PossAct(n, s)$, are added to the new belief state. Because non-executable actions result in situations being discarded, the 'probability' distribution over all the situations in the new belief state may not sum to 1; the distribution thus needs to be normalized.

$senseCond$ is mentioned in the definition of $BestDoObserve$. It is similar to the the definition in Section 2, only, here it is defined for observations instead of actions.

$BestDoPO$ is recursively called with the remaining program and with the horizon h decremented by 1. Also note that the recursive $BestDoPO$ will now operate with the updated belief b'. In the following definition, $\{o_k\}$ is a single (remaining) observation in the set returned by $choiceObs'$.

Observations possible

$$\begin{aligned}
&BestDoObserve(\{o_k\}, a.rest, b, h, \pi, v, pr) \doteq \\
&\quad \neg PossObs(o_k, a, b) \wedge \pi = Stop \wedge v = 0 \wedge pr = 0 \vee \\
&\quad PossObs(o_k, a, b) \wedge b' = BU(o_k, a, b) \wedge \\
&\quad \exists \pi', v', pr'.BestDoPo(rest, b', h-1, \pi', v', pr') \wedge \\
&\quad senseCond(o_k, \varphi_k) \wedge \pi = \varphi_k?; \pi' \wedge \\
&\quad v = v' \cdot belObs(o_k, a, b) \wedge pr = pr' \cdot belObs(o_k, a, b).
\end{aligned}$$

When the set of observations has more than one observation in it, the formula definition is slightly different, but similar to the one above: the first branch of possible observations is processed, and the other branches in the remainder of the set are processed recursively.

When the planning horizon has reached zero or when all actions have been 'performed' (no remaining actions in the input program), there will be no further recursive calls.

Conditional statement and test action formulas are similar to those of Golog, except that the 'condition' or 'test statement' respectively, are with respect to the agent's current belief state, and probabilities involved in these formulas are influenced in proportion to the agent's *degree of belief* [8] in the respective statements (see [10] for details). Sequential composition and conditional iteration are defined as one would expect according to complex actions in Golog.

6. A Simple Example

A very simple example follows to illustrate how $BestDoPO$ calculates an optimal policy. We use a four-state world as depicted in Figure 3. The agent's initial belief state is $b_0 = \{(s1, 0.04), (s2, 0.95), (s3, 0.0), (s4, 0.01)\}$. The only actions available to the agent are $left$ and $right$. We define the actions' stochasticity with $\forall n, a, s.choiceNat(n, a, s) \equiv TRUE$, with associated probabilities:

$$probNat(left, left, s) = probNat(right, right, s) = 0.9$$
$$probNat(right, left, s) = probNat(left, right, s) = 0.1$$

The probability that any of the actions will cause an observation of nothing ($obsnil$) is 1: $probObs(obsnil, a, s) = p \equiv (a = left \vee a = right) \wedge p = 1$. The corresponding definition for choice of observations is $choiceObs(obsnil, a, s) \equiv (a = left \vee a = right)$.

Let the fluent $At(loc(x), s)$ denote the location of the agent. It's successor-state axiom is defined by

$$\begin{aligned}
&At(loc(x), do(a, s)) \equiv \\
&\quad a = left \wedge (At(loc(x+1), s) \wedge x \neq 1) \\
&\qquad \vee (At(loc(x), s) \wedge x = 1) \vee \\
&\quad a = right \wedge (At(loc(x-1), s) \wedge x \neq 4) \\
&\qquad \vee (At(loc(x), s) \wedge x = 4) \vee \\
&\quad At(loc(x), s) \wedge (a \neq left \wedge a \neq right).
\end{aligned}$$

For simplicity, we allow all actions and observations all of the time, that is, $\forall a, s.PossAct(a, s) \equiv TRUE$ and $\forall a, s.PossObs(a, s) \equiv TRUE$.

Finally, we specify the sensing condition predicate and the reward function. $senseCond(obsnil, \psi) \equiv \psi = OutcomeIs(nil, sensor\_value))$ with $OutcomeIs(obsnil, sensor\_value) \equiv TRUE$ and $reward(s) =$ **if** $At(loc(3), s)$ **then** 1 **else** -1; hence, the agent's goal should be location 3.

Assume, the agent is equipped with the following program; an initial input for $BestDoPO$:

$$\begin{aligned}
&BestDoPO(\textbf{while}\ (true\ \textbf{do}\ [left \mid right]), \\
&\quad \{(s1, 0.04), (s2, 0.95), (s3, 0.0), (s4, 0.01)\}, 1, \pi, v, pr);
\end{aligned}$$

the algorithm must computing a one-step optimal policy.

After the iterative component of the program is processed, the following call is made, as per the definition of $BestDoPO$ for the nondeterministic choice of actions:

$$\begin{aligned}
&BestDoPO([left \mid right]; rest, b_0, 1, \pi, v, pr) \\
&\quad \exists \pi_1, v_1, pr_1.BestDoPO(left; rest, b_0, 1, \pi_1, v_1, pr_1) \wedge \\
&\quad \exists \pi_2, v_2, pr_2.BestDoPO(right; rest, b_0, 1, \pi_2, v_2, pr_2) \wedge \\
&\quad ((v_1, left) \geq (v_2, right) \wedge \pi = \pi_1 \wedge v = v_1 \wedge pr = pr_1) \vee \\
&\quad (v_1, left) < (v_2, right) \wedge \pi = \pi_2 \wedge v = v_2 \wedge pr = pr_2)),
\end{aligned}$$

where $rest$ is **while** $(true\ \textbf{do}\ [left \mid right])$. Then the recursive $BestDoPO$s make use of the "Probabilistic observation" definition of the formula. Because–by the action precondition axioms for this example–$left$ and $right$ are always executable, the following portion (times two) of the formula are applicable:

$$\exists \pi', v'.BestDoObserve(choiceObs'(left), \quad (4)$$
$$left, rest, b_0, 1, \pi', v', pr) \wedge \quad (5)$$
$$\pi = left; \pi' \wedge v = R(b_0) + v' \quad (6)$$

and
$$\exists \pi', v'.BestDoObserve(choiceObs'(right), \quad (7)$$
$$right, rest, b_0, 1, \pi', v', pr) \wedge \quad (8)$$
$$\pi = right; \pi' \wedge v = R(b_0) + v'. \quad (9)$$

For Lines (4) and (5) the following portion of the "Observations possible" definition is applicable:

$$b' = BU(obsnil, left, b_0) \wedge$$
$$\neg\pi', v', pr'.\, BestDoPO(rest, b', 1-1, \pi', v', pr') \wedge$$
$$senseCond(obsnil, \phi) \wedge \pi = \phi?; \pi' \wedge$$
$$v = v' \cdot belObs(obsnil, left, b_0) \wedge$$
$$pr = pr' \cdot belObs(obsnil, left, b_0).$$

In this formula (portion), ϕ unifies with $OutcomeIs(obsnil, sensor\_value)$ and because the recursive call to $BestDoPO$ has a zero horizon, $\pi' = nil$, and thus $\pi = (OutcomeIs(obsnil, sensor\_value))?; nil$.

The updated belief is an input to a 'zero horizon' call and will therefore be used to determine v'; we calculate the new belief state $b' = BU(obsnil, left, b_0)$ now (we work out only the first new element of b' in detail):

$$(s^+, p^+) \in b_{temp} : s^+ = do(left, s_1) \wedge p^+ = 0.04 \times 1 \times 0.9.$$

Because all actions are possible, the only effect that normalization (in the update function) has, is to remove $(do(left, s3), 0.0)$ and $(do(right, s3), 0.0)$ from the new belief state, because of their zero probabilities. $BU(obsnil, left, b_0)$ results in

$$b' = \{(do(left, s1), 0.036), (do(right, s1), 0.004),$$
$$(do(left, s2), 0.855), (do(right, s2), 0.095),$$
$$(do(left, s4), 0.009), (do(right, s4), 0.001)\}.$$

$belObs(obsnil, left, b_0) = (0.04)(1)+(0.95)(1)+(0.0)(1)+(0.01)(1) = 1$ and hence $v = v' \times 1$, and $pr = pr' \times 1$. Due to the 'zero horizon' call, $v' = R(b') = (-1)(.036) + (-1)(.004) + (-1)(.885) + (1)(.095) + (1)(.009) + (-1)(.001) = -0.822$ and $pr' = 1.0$. Therefore, $v = -0.822$, and $pr = 1.0$.

Now we can instantiate Line (6) as follows: $\pi = left; OutcomeIs(obsnil, sensor\_value))?; nil \wedge v = -1 + (-0.822)$. Similarly, we can instantiate Line (9) as $\pi = right; OutcomeIs(obsnil, sensor\_value))?; nil \wedge v = -1 + (0.712)$.

Then finally, we find that $((-0.822, left) < (0.712, right)$ and return the policy $\pi = right; OutcomeIs(obsnil, sensor\_value))?; nil$, with total expected reward $v = -0.288$ and program success probability $pr = 1$.

Note that for the sake of clarity, we assumed noise-free perceptions. It should be clear though, that our algorithm can deal with noisy perceptions as well.

Considering that the agent believed to a relatively high degree that it was initially just left of the 'high-reward' location, and given that its observations are complete and its actions are not extremely erroneous, we would expect the agent's first move to be rightwards, as indeed, the policy recommends.

7. Discussion and Conclusion

In this paper we have given a formal semantics for an action planner that can generate control policies for agents in partially observable domains. The language we used for the specification is the agent programming language DTGolog. Much of the semantics is similar to [10]. Their approach is however not for a single-agent domain.

An example was presented that showed in detail the processes involved in generating a policy for an agent with probabilistic beliefs in a partially observable and stochastic domain.

We implemented the POMDP planner in ECL^iPS^e *Prolog*. The implementation was set up for two toy worlds: a four-state world where the states are all in a row, and a five-by-five grid world. In both cases, an agent must find a 'star'. Preliminary experiments with the implementation showed the potential for practical application of the planner presented in this paper: the results of the experiments showed that the policies generated are reasonable, and overall, the planner seems to work correctly. However, benchmarking and comparison to other similar planners (for problems in similarly stochastic and noisy domains) still needs to be conducted.

8. References

[1] Levesque, H., Reiter, R., Lespérance, Y., Lin, F., and Scherl, R., "GOLOG: A Logic programming language for dynamic domain", *Journal of Logic Programming*, 31:59–84, 1997.

[2] Boutilier, C., Reiter, R., Soutchanski, M., and Thrun, S., "Decision-theoretic, high-level agent programming in the situation calculus", in *Proceedings AAAI-2000*, 2000, pp. 355–362.

[3] Reiter, R., *Knowledge in action: logical foundations for specifying and implementing dynamical systems*, Massachusetts/England: MIT Press, 2001.

[4] Brachman, R. J. and Levesque, H. J., *Knowledge representation and reasoning*, California: Morgan Kaufmann, 2004.

[5] Kaelbling, L. P., Littman, M. L., and Cassandra, A. R., "Planning and acting in partially observable stochastic domains", *Artificial Intelligence*, 101(1–2):99–134, 1998.

[6] Pineau, J., *Tractable planning under uncertainty: exploiting structure*, Robotics Institute, Carnegie Mellon University, 2004. Unpublished doctoral dissertation.

[7] Clemen, R. T., and Reilly, T., *Making hard decisions*, California: Duxbury, 2001.

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

[9] Grosskreutz, H., *Towards more realistic logic-based robot controllers in the Golog framework*, Knowledge-Based Systems Group, Rheinisch-Westfälischen Technischen Hochschule, 2002.

[10] Finzi, A, and Lukasiewicz, T., "Game-theoretic agent programming in Golog under partial observability", in *KI 2006: Advances in Artificial Intelligence*, 2007, pp. 113–127.

[11] Ferrein, A. and Lakemeyer G. "Logic-based robot control in highly dynamic domains.", *Journal of Robotics and Autonomous Systems, Special Issue on Semantic Knowledge in Robotics* 2008. to appear.

[12] Bonet, B. and Geffner, H., "Planning and control in artificial intelligence: a unifying perspective", *Applied Intelligence*, 14(3): 237–252, 2001.

[13] Poole, D., "Planning and acting in partially observable stochastic domains", *Linköping Electronic Articles in Computer and Information Science*, 3(8), 1998.