An On-line Decision-Theoretic Golog Interpreter.

Mikhail Soutchanski
Dept. of Computer Science
University of Toronto
Toronto, ON M5S 3H5
mes@cs.toronto.edu

Abstract

We consider an on-line decision-theoretic interpreter and incremental execution of Golog programs. This new interpreter is intended to overcome some limitations of the off-line interpreter proposed in [Boutilier *et al.*, 2000]. We introduce two new search control operators that can be mentioned in Golog programs: the on-line interpreter takes advantage of one of them to save computational efforts. In addition to sensing actions designed to identify outcomes of stochastic actions, we consider a new representation for sensing actions that may return both binary and real valued data at the run time. Programmers may use sensing actions explicitly in Golog programs whenever results of sensing are required to evaluate tests. The representation for sensing actions that we introduce allows the use of regression, a computationally efficient mechanism for evaluation of tests. We describe an implementation of the on-line incremental decision-theoretic Golog interpreter in Prolog. The implementation was successfully tested on the B21 robot manufactured by RWI.

1 Introduction

The aim of this paper is to provide a new on-line architecture of designing controllers for autonomous agents. Specifically, we explore controllers for mobile robots programmed in DTGolog, an extension of the high-level programming language Golog. DTGolog aims to provide a seamless integration of a decision-theoretic planner based on Markov decision processes with an expressive set of program control structures available in Golog. The motivation for this research is provided in [Boutilier *et al.*, 2000], and we ask the reader to consult that paper for additional technical details and arguments why neither model-based planning, nor programming alone can manage the conceptual complexity of devising controllers for mobile robots. The DTGolog interpreter described in [Boutilier *et al.*, 2000] is an off-line interpreter that computes the optimal conditional policy π, the probability pr that π can be executed successfully and the expected utility u of the policy. The semantics of a DTGolog program p is defined by the predicate $BestDo(p, s, h, \pi, u, pr)$, where s is a starting situation and h is a given finite horizon. The policy π returned by the off-line interpreter is a Golog program consisting of the sequential composition of agent actions, *senseEffect*(a) sensing actions (which serve to identify a real outcome of a stochastic action a) and conditionals (**if** ϕ **then** π_1 **else** π_2), where ϕ is a situation calculus formula that provides a test condition to decide which branch of a policy the agent should take given the result of sensing. The interpreter looks ahead up to the last choice point in each branch of the program (say, between alternative primitive actions), makes the choice and then proceeds backwards recursively, deciding at each choice point what branch is optimal. It is assumed that once the off-line DTGolog interpreter has computed an optimal policy, the policy is given to the robotics software to control a real mobile robot.

However this off-line architecture has a number of limitations. Imagine that we are interested in executing a program $(p_1; p_2)$ where both sub-programs p_1 and p_2 are very large nondeterministic DTGolog programs designed to solve 'independent' decision-theoretic problems: p_2 is supposed to start in a certain set of states, but from the perspective of p_2 it is not important what policy will be used to reach one of those states. Intuitively, in this case we are interested in computing first an optimal policy π_1 (that corresponds to p_1), executing π_1 in the real world and then computing and executing an optimal policy π_2. But the off-line interpreter can return only the optimal policy π that corresponds to the whole program $(p_1; p_2)$, spending on this computation more time than necessary: to compute and execute π_2 it is not relevant what decisions have to be made during the execution of π_1. The second limitation becomes evident if in addition to *senseEffect* sensing actions serving to identify outcomes of stochastic actions, we need to explicitly include sensing actions in the program (and those sensing actions cannot be characterized as stochastic actions with a fixed finite set of outcomes). Imagine that we are given a program

$p_1; (\pi v, t).[(now(t))?; sense(Q, v, t);$ **if** ϕ **then** p_2 **else** $p_3]$,

where $sense(Q, v, t)$ is a sensing action that returns at time t a measurement v of a quantity Q (e.g., the current coordinates, the battery voltage) and the condition ϕ depends on the real data v that will be returned by sensors (in the program above p_1, p_2, p_3 are sub-programs, the choice operator π binds variables v and t and the test $now(t)?$ grounds the current time). Intuitively, we want to compute an optimal policy π_1 (corresponding to p_1) off-line, execute π_1 in the real world, sense v and then compute and execute an optimal policy π_2 that corresponds to the conditional Golog program in brackets. But the off-line interpreter is not able to compute an optimal policy if a given program includes explicit sensing actions and no information is available about the possible results of sensing. Note that the nondeterministic choice operator π (it occurs in front of the program) should not be confused with policies π_1 and π_2 computed by an interpreter.

We propose to compute and execute optimal policies on-line using a new incremental decision theoretic interpreter. It works in a step-by-step mode. Given a Golog program p and a starting situation s, it computes an optimal policy π and the program p' that remains when a first action a in π will be executed. At the next stage, the action a is executed in the real world. The interpreter gets sensory information to identify which outcome of a has actually occurred if a is a stochastic action: this may require doing a sequence of sensing actions on-line. The action a (and possibly sensing actions performed after that) results in a new situation. Then the cycle of computing an optimal policy and remaining program, executing the first action and getting sensory information (if necessary) repeats until the program terminates or execution fails. In the context of the incremental on-line execution, we define a new programming operator $optimize(p)$ that limits the scope of search performed by the interpreter. If p is the whole program, then no computational efforts are saved when the interpreter computes an optimal policy from p, but if the programmer writes $optimize(p_1); p_2$, then the on-line incremental interpreter will compute and execute step-by-step the Golog program p_1 without looking ahead to decisions that need to be made in p_2. If the programmer knows that the sensing action $sense(Q, v, t)$ is necessary to evaluate the condition ϕ, then using the program

$$optimize(p_1); (\pi t, v).[(now(t))?; optimize(sense(Q, v, t)); \textbf{if } \phi \textbf{ then } p_2 \textbf{ else } p_3]$$

the required information about Q will be obtained before the incremental interpreter will proceed to the execution of the conditional. If the sensing action $sense(Q, v, t)$ has many different outcomes, then this approach gives computational advantages over the off-line approach to computing an optimal policy.

Thus, the incremental interpretation of the decision-theoretic Golog programs needs an account of sensing (formulated in the situation calculus) that will satisfy several criteria which naturally arise in the robotics context. There are several accounts of sensing in the situation calculus that address different aspects of reasoning about sensory and physical actions [Bacchus *et al.*, 1995; De Giacomo and Levesque, 1999a; 1999b; Funge, 1998; Grosskreutz, 2000; Lakemeyer, 1999; Levesque, 1996; Pirri and Finzi, 1999; Scherl and Levesque, 1993].

Below we propose a new representation of sensing that simplifies reasoning about results of sensing, does not require consistency of sensory information with the domain theory, leads to a natural and sound implementation and has connections with representation of knowledge and sensing considered in [Reiter, 2000].

In Section 2 we recall the representation of the decision theoretic domain introduced in [Boutilier *et al.*, 2000]. In Section 3 we propose a representation for sensing actions and consider several examples. In section 4 we consider the on-line incremental decision-theoretic interpreter. Section 5 discusses connections with previously published papers.

2 The decision theoretic problem representation

The paper [Boutilier *et al.*, 2000] introduces the representation of problem domains that do not include sensing actions. The representation is based on the distinction between agent actions (which can be either deterministic or stochastic) and nature's actions which correspond to separate outcomes of a stochastic agent action. Nature's actions are considered deterministic. They cannot be executed by the agent itself, therefore they never occur in policies which the agent executes. When the agent does a stochastic action a in a situation s, nature chooses one of the outcomes n of that action and the situation $do(n, s)$ is considered as one of resulting situations. In accordance with this perspective, the evolution of a stochastic transition system is specified by precondition and successor state axioms which never mention stochastic agent actions, but mention only deterministic agent actions and nature's actions. In [Boutilier *et al.*, 2000], it is suggested to characterize the DTGolog problem domain by: 1) the predicate $agentAction(a)$ which holds if a is an agent action, 2) the predicate $stochastic(a, s, n)$ meaning that nature's action n is one of outcomes of the stochastic agent action a in the situation s, 3) the function symbol $prob(n, s)$ that denotes the probability of nature's action n in situation s, and 4) the predicate $senseCond(n, \phi)$ specifying the test condition ϕ serving to identify the outcome n of the last stochastic action, 5) the function symbol $reward(do(a, s))$ denotes rewards and costs as functions of the current situation $do(a, s)$, the action a or both.

As an example, imagine a robot moving between different locations: the process of going is initiated by a deterministic action $startGo(l_1, l_2, t)$ but is terminated by a stochastic action $endGo(l_1, l_2, t)$ that may have two different outcomes: successful arrival $endGoS(l_1, l_2, t)$ and unsuccessful stop in a place different from the destination $endGoF(l_1, l_3, t)$ (the robot gets stuck in the hall or cannot enter an office because its door is closed). We represent the process of moving between locations l_1 and l_2 by the relational fluent $going(l_1, l_2, s)$ and represent a (symbolic) location of the robot by the relational fluent $robotLoc(l, s)$ meaning that l (the office of an employee, the hall or the main office) is the place where the robot is. The transitions in the stochastic dynamical system describing the robot's motion are characterized by the following precondition and successor state axioms.

$$Poss(startGo(l_1, l_2, t), s) \equiv \neg(\exists l, l') going(l, l', s) \wedge robotLoc(l_1, s),$$

$$Poss(endGoS(l_1, l_2, t), s) \equiv going(l_1, l_2, s),$$
$$Poss(endGoF(l_1, l_2, t), s) \equiv \exists l'.going(l_1, l', s) \wedge l' \neq l_2,$$

$$\begin{aligned} going(l, l', do(a, s)) \equiv\ & (\exists t) a = startGo(l, l', t) \vee \\ & going(l, l', s) \wedge \neg(\exists t) a = endGoS(l, l', t) \vee \\ & going(l, l', s) \wedge \neg(\exists t, l'') a = endGoF(l, l'', t). \end{aligned}$$

The real outcome n of a stochastic agent action a can be identified only from sensory information. This information has to be obtained by executing sensing actions. In the following section we propose a new representation for sensing actions providing a seamless integration with the representation considered in this section.

3 Sensing actions

In contrast to physical actions, sensing actions do not change any properties of the external world, but return values measured by sensors. Despite this difference, it is convenient to treat both physical and sensing actions equally in successor state axioms. This approach is justifiable if fluents represent what the robot 'knows' about the world (see Figure 1). More specifically, let the high-level control module of the robot be provided with the internal logical model of the world (the set of precondition and successor state axioms) and the axiomatization of the initial situation. The programmer expresses in this axiomatization his (incomplete) knowledge of the initial situation and captures certain important effects of the robot's actions, but some other effects and actions of other agents may remain unmodeled. When the robot does an action in the real world (i.e., the high-level control module sends a command to effectors and effectors execute this command), it does not know directly and immediately what effects in reality this action will produce: the high-level control module may only compute expected effects using the internal logical model. Similarly, if the robot is informed about actions executed by external agents, the high-level control module may compute expected effects of those exogenous actions (if axioms account for them). Thus, from the point of view of the programmer who designed the axioms, the high-level control module maintains the set of beliefs that the robot has about the real world. This set of beliefs needs feedback from the real world because of possible contingencies, incompleteness of the initial information and unobserved or unmodeled actions executed by other agents. To gain the feedback, the high-level control module requests information from sensors and they return required data. We find it convenient to represent information gathered from sensors by an argument of the sensing action: a value of the argument will be instantiated at the run time when the sensing action will be executed. Then, all the high-level control module needs to know is the current situation (action log): expected effects of actions on fluents can be computed from the given sequence of physical and sensing actions.

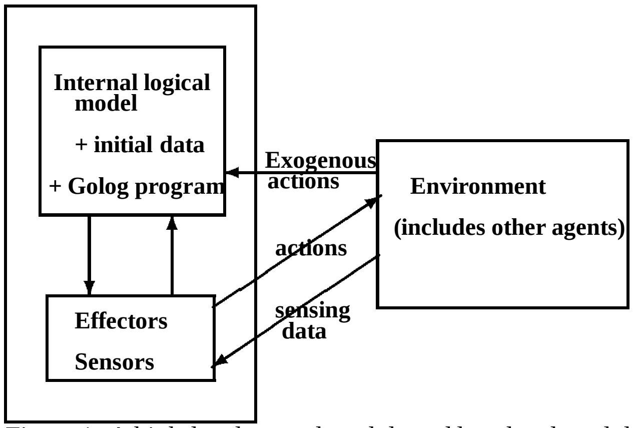

Figure 1: A high-level control module and low-level modules interacting with the environment.

In the sequel, we consider only deterministic sensing actions, but noisy sensing actions can be represented as well.

We suggest representing sensing actions by the functional symbol $sense(q, v, t)$ where q is what to sense, v is term representing a run time value returned by the sensors and t is time; the predicate $senseAction(a)$ is true whenever a is a sensing action. We proceed to consider several examples of successor state axioms that employ our representation.

1. Let $sense(Coord, l, t)$ be the sensing action that returns the pair $l = (x, y)$ of geometric coordinates of the current robot location on the two-dimensional grid and $gridLoc(x, y, s)$ be a relational fluent that is true if (x, y) are the coordinates of the robot's location in s. In this example we assume that all actions are deterministic (we do this only to simplify the exposition of this example). The process of moving (represented by the relational fluent $moving(x_1, y_1, x_2, y_2, s)$) from the grid location (x_1, y_1) to the point (x_2, y_2) is initiated by the deterministic instantaneous action $startMove(x_1, y_1, x_2, y_2, t)$ and is terminated by the deterministic action $endMove(x_1, y_1, x_2, y_2, t)$. The robot may also change its location if it is transported by an external agent from one place to another: the exogenous actions $take(x_1, y_1, t)$ and $put(x_2, y_2, t)$ account for this (and the fluent $transported(s)$ represents the process of moving the robot by an external agent). The following successor state and precondition axioms characterize aforementioned fluents and actions.

$$\begin{array}{l}
gridLoc(x, y, do(a, s)) \equiv \\
((\exists t, l) a = sense(Coord, l, t) \wedge xCrd(l) = x \wedge yCrd(l) = y) \vee \\
(\neg transported(s) \wedge (\exists t, x', y') a = endMove(x', y', x, y, t)) \vee \\
\quad (transported(s) \wedge (\exists t) a = put(x, y, t)) \vee \\
\quad\quad gridLoc(x, y, s) \wedge \neg(\exists t, x', y') a = put(x', y', t) \wedge \\
\quad\quad\quad \neg(\exists l, t) a = sense(Coord, l, t) \wedge \\
\quad\quad\quad\quad \neg(\exists x, y, x', y', t) a = endMove(x, y, x', y', t),
\end{array}$$

where $xCrd(l), yCrd(l)$ denote, respectively, x and y components of the current robot location l.

$$\begin{array}{l}
moving(x_1, y_1, x_2, y_2, do(a, s)) \equiv \\
\quad (\exists t) a = startMove(x_1, y_1, x_2, y_2, t) \vee \\
\quad\quad moving(x_1, y_1, x_2, y_2, s) \wedge \\
\quad\quad\quad \neg(\exists t) a = endMove(x_1, y_1, x_2, y_2, t).
\end{array}$$

$$\begin{array}{l}
transported(do(a, s)) \equiv (\exists t, x, y)\, a = take(x, y, t) \vee \\
\quad\quad transported(s) \wedge \neg(\exists t, x', y') a = put(x', y', t) \vee \\
(\neg \exists x, y, x', y') moving(x, y, x', y') \wedge gridLoc(x, y, s) \wedge \\
\quad (\exists t, l) a = sense(Coord, l, t) \wedge (xCrd(l) \neq x \vee yCrd(l) \neq y).
\end{array}$$

$$\begin{array}{l}
Poss(startMove(x_1, y_1, x_2, y_2, t), s) \equiv \neg transported(s) \wedge \\
\quad \neg(\exists x, y, x', y')\, moving(x, y, x', y', s) \wedge gridLoc(x_1, y_1, s), \\
Poss(endMove(x_1, y_1, x_2, y_2, t), s) \equiv moving(x_1, y_1, x_2, y_2, s),
\end{array}$$

$$\begin{array}{l}
Poss(take(x, y, t), s) \equiv \neg transported(s) \wedge gridLoc(x, y, s) \wedge \\
\quad\quad \neg(\exists x_1, y_1, x_2, y_2)\, moving(x_1, y_1, x_2, y_2, s), \\
\quad Poss(put(x, y, t), s) \equiv transported(s).
\end{array}$$

Imagine that in the initial situation the robot stays at (0,0); later, at time 1, it starts moving from (0,0) to (2,3), but when the robot stops at time 11 and senses its coordinates at time 12, its sensors tell it that it is located at (4,4). The sensory information is inconsistent with the expected location of the robot, but this discrepancy can be attributed to unobserved actions of an external agent who transported the robot. Hence, the final situation is not

$$\begin{array}{l}
S_3 = do(sense(Coord, (4,4), 12), do(endMove(0,0,2,3,11), \\
\quad\quad do(startMove(0,0,2,3,1), S_0))),
\end{array}$$

as we originally expected, but the situation resulting from the execution of exogenous actions $take(2, 3, T_1)$ and $put(4, 4, T_2)$, in the situation when the robot ends moving, followed by the sensing action. The exogenous actions occurred at unknown times T_1, T_2 (we may say about the actual history only that $11 \leq T_1 < T_2 \leq 12$).[1]

2. The robot can determine its location using data from sonar and laser sensors. But if the last action was not sensing coordinates (x, y), then the current location can be determined

[1]In the sequel, we do not consider how the discrepancy between results of a deterministic action and observations obtained from sensors can be explained by occurrences of unobserved exogenous actions. However, our example indicates that inconsistency between physical and sensory actions can be resolved by solving a corresponding diagnostic problem (e.g., see [McIlraith, 1998]).

from the previous location and the actions that the robot has executed. The process of going from l to l' is initiated by the deterministic action $startGo(l, l', t)$ and is terminated by the stochastic action $endGo(l, l', t)$ (axiomatized in Section 2).

$$\begin{aligned}
&robotLoc(l, do(a, s)) \equiv (\exists t, v, x, y)\, a = sense(Coord, v, t) \\
&\quad \wedge xCrd(v) = x \wedge yCrd(v) = y \wedge \\
&\quad ((\exists p)\, l = \mathit{office}(p) \wedge \mathit{inOffice}(l, x, y) \vee \\
&\qquad l = Hall \wedge \neg(\exists p)\, \mathit{inOffice}(\mathit{office}(p), x, y)) \vee \\
&(\exists t, l_1) a = endGoS(l_1, l, t) \vee (\exists t, l_1) a = endGoF(l_1, l, t) \vee \\
&robotLoc(l, s) \wedge \neg(\exists t, l_2, l_3, v)\, (a = sense(Coord, v, t) \wedge \\
&\qquad a = endGoS(l_2, l_3, t) \wedge a = endGoF(l_2, l_3, t)).
\end{aligned}$$

where the predicate *inOffice*(l, x, y) is true if the pair (x, y) is inside of the office l, the functional symbols *bottomY*(l), *topY*(l), *rightX*(l) and *leftX*(l) represent coordinates of top left and bottom right corners of a rectangle that contains the office l inside: When the robot stops and senses coordinates, it determines its real location and the high-level control module can identify the outcome of the stochastic action $endGo(l, l', t)$: whether the robot stopped successfully or failed to arrive at the intended destination.

3. Let $give(item, person, t)$ be a stochastic action that has two different outcomes: $giveS(item, person, t)$ – the robot gives successfully an $item$ to $person$ at time t – and $giveF(item, person, t)$ – the action of giving an $item$ to $person$ is unsuccessful. Let $sense(Ackn, v, t)$ be the action of sensing whether delivery of the $item$ to $person$ was successful or not: if it was, then delivery is acknowledged and v=1, if not – the result of sensing is v=0. The following successor state axiom characterizes how the fluent *hasCoffee*$(person, s)$ changes from situation to situation: whenever the robot is in the office of $person$ and it senses that one of its buttons was pressed, it is assumed that the $person$ pressed a button to acknowledge that she has a coffee. From this sensory information, the high-level control module can identify whether the outcome of $give(item, person, t)$ was successful or not.

$$\begin{aligned}
&\mathit{hasCoffee}(pers, do(a, s)) \equiv \\
&\quad (\exists t) a = giveS(\mathit{Coffee}, pers, t) \vee \mathit{hasCoffee}(pers, s) \vee \\
&\quad (\exists t, v) a = sense(Ackn, v, t) \wedge v = 1 \wedge \\
&\qquad robotLoc(\mathit{office}(pers), s).
\end{aligned}$$

It is surprisingly straightforward to use regression in our setting to solve the forward projection task. Let $\mathcal{D}$ be a background axiomatization of the domain (a set of successor state axioms, precondition axioms, unique name axioms and a set of first order sentences whose only situation term is S_0) and let $\phi(s)$ be a situation calculus formula that has the free variable s as the only situational argument. Suppose we are given a ground situational term S that may mention both physical and sensing actions. The forward projection task is to determine whether

$$\mathcal{D} \models \phi(S)$$

Regression is a computationally efficient way of solving the forward projection task [Reiter, 2000; Pirri and Reiter, 1999] when S does not mention any sensing actions. The representation introduced above allows us to use regression also in the case when S mentions sensing actions explicitly. Thus, we can use regression to evaluate tests $(\phi)?$ in Golog programs with sensing actions: no modifications are required. In addition, our approach allows us to use an implementation in Prolog considered in [Reiter, 2000]. There is also an interesting connection between our representation of 'beliefs' and sensing and the approach to knowledge-based programming [Reiter, 2000]. In [Soutchanski, 2001] we show that his approach and our approach to the solution of the forward projection task (with sensing actions) are equivalent.

4 The incremental on-line DTGolog interpreter

The incremental DTGolog interpreter uses the predicate *IncrBestDo*$(p_1, s, p_2, h, \pi, u, pr)$ and the predicate $Final(p, s, \pi, u)$. The former predicate takes as input the Golog program p_1, starting situation s, horizon h and returns the optimal conditional policy π, its expected utility u, the probability of success pr, and the program p_2 that remains after executing the first action from the policy π. The latter predicate tells when the execution of the policy completes: $Final(p, s, \pi, u)$ is true if either the program p is Nil (the null program) or $Stop$ (a zero cost action that takes the agent to an absorbing state meaning that the execution failed), or if the policy π is Nil or $Stop$. In all these cases, u is simply the reward associated with the situation s. Note that in comparison to *BestDo*, *IncrBestDo* has an additional argument p_2 representing the program that remains to be executed.

IncrBestDo$(p_1, s, p_2, h, \pi, u, pr)$ is defined inductively on the structure of a Golog program p_1. Below we consider its definition in the case when the program p_1 begins with a deterministic agent action.

$$\begin{aligned}
&\mathit{IncrBestDo}(a; p_1, s, p_2, h, \pi, u, pr) \overset{def}{=} \\
&\quad [\neg Poss(a, s) \wedge \\
&\qquad p_2 = Nil \wedge \pi = \mathit{Stop} \wedge pr = 0 \wedge u = \mathit{reward}(s) \vee \\
&\quad Poss(a, s) \wedge p_2 = p_1 \ \wedge\ \exists(p', \pi', u', pr') \\
&\qquad \mathit{IncrBestDo}(p_2, do(a, s), p', h{-}1, \pi', u', pr') \wedge \\
&\qquad\quad \pi = (a; \pi') \wedge u = reward(s) + u' \wedge pr = pr'].
\end{aligned}$$

If a deterministic agent action a is possible in situation s, then we compute the optimal policy π' of the remaining program p_1, its expected utility u' and the probability of successful termination pr'. Because a is a deterministic action, the probability pr' that the policy π' will complete successfully is the same as for the program itself; the expected utility u is a sum of a reward for being in s and the expected utility of continuation u'. If a is not possible, then the remaining program is Nil, and the policy is the *Stop* action, the expected utility is the reward in s and the probability of success is 0.

Other cases are defined similarly to $BestDo$, e.g., if the program begins with the finite nondeterministic choice $(\pi(x : \tau)p); p'$, where τ is the finite set $\{c_1, \ldots, c_n\}$ and the choice binds all free occurrences of x in p to one of the elements:

$$\begin{aligned}
&\mathit{IncrBestDo}((\pi(x : \tau)p); p', s, p_r, h, pol, u, pr) \overset{def}{=} \\
&\quad \mathit{IncrBestDo}((p|^x_{c1} \ | \ \ldots \ | \ p|^x_{cn}); p', s, p_r, h, pol, u, pr)
\end{aligned}$$

where $p|^x_c$ means substitution of c for all free occurrences of x in p. Thus, the optimal policy pol corresponds to the element c in τ that delivers the best execution. Note that the remaining program p_r is the same on the both sides of the definition.

Recall that policies are Golog programs as well. Moreover, if p is a Golog program that contains many nondeterministic choices, the optimal policy π computed from p is a conditional program that does not involve any nondeterministic choices. This observation suggests that programmers may wish to take advantage of the structure in a decision theoretic problem and use explicit search control operators that limit bounds where the search for an optimal policy has to concentrate. In addition to the standard set of Golog programming constructs, we introduce two new operators: $local(p)$ and $optimize(p)$. Intuitively, the program $local(p_1); p_2$ means the following. First, compute the optimal policy π_1 corresponding to the sub-program p_1, then compute the optimal policy π corresponding to the program $\pi_1; p_2$. If both sub-programs p_1 and p_2 are highly nondeterministic, then using the operator $local(p_1)$ the programmer indicates where the computational efforts can be saved: there is no need in looking ahead further than p_1 to compute π_1. Thus, in the case that a Golog program begins with $local(p)$

$$\begin{aligned}&\textit{IncrBestDo}(local(p_1); p_2, s, p_r, h, \pi, u, pr) \overset{def}{=}\\ &\quad (\exists p, \pi_1, u_1, pr_1)\textit{IncrBestDo}(p_1; Nil, s, p, h, \pi_1, u_1, pr_1) \wedge\\ &\qquad \textit{IncrBestDo}(\pi_1; p_2, s, p_r, h, \pi, u, pr).\end{aligned}$$

The construct $local(p_1)$ limits the search, but once the policy π_1 was computed and the first action in π_1 was executed, the remaining part of the program has no indication where the search may concentrate. For this reason, the programmer may find convenient to use another search control operator that once used persists in the remaining part of the program until the program inside the scopes of that operator terminates. This operator is called $optimize(p)$ and is specified by the following abbreviation.

$$\begin{aligned}&\textit{IncrBestDo}(optimize(p_1); p_2, s, p_r, h, \pi, u, pr) \overset{def}{=}\\ &\quad (\exists p')\ \textit{IncrBestDo}(p_1; Nil, s, p', h, \pi, u, pr) \wedge\\ &\quad (\ p' \neq Nil \wedge p_r = (optimize(p'); p_2) \vee\\ &\qquad p' = Nil \wedge p_r = p_2\ ).\end{aligned}$$

According to this specification, an optimal policy π can be computed without looking ahead to the program p_2; hence, using $optimize(p_1)$ a programmer can express a domain specific procedural knowledge to save computational efforts. Note that when p' is *Nil*, i.e. there will be nothing in p_1 to execute after doing the only action in π, the remaining program p_r contains only p_2.

4.1 Implementing the on-line interpreter

Given the definitions of *IncrBestDo* mentioned in the previous sub-section, we can consider now the on-line interpretation coupled with execution of Golog programs. The definitions of *IncrBestDo*$(p_1, s, p_2, h, \pi, u, pr)$ translate directly into Prolog clauses (we omit them here). The on-line interpreter calls the off-line *incrBestDo(E,S,ER,H,Pol1,U1,Prob1)* interpreter to compute an optimal policy from the given program expression E, gets the first action of the optimal policy, commits to it, does it in the physical world, then repeats with the rest of the program. The following is such an interpreter implemented in Prolog:

```
online(E,S,H,Pol,U) :-
incrBestDo(E,S,ER,H,Pol1,U1,Prob1),
( final(ER,S,H,Pol1,U1), Pol=Pol1, U=U1 ;
  reward(R,S),
  Pol1 = (A : Rest),
 ( agentAction(A), not stochastic(A,S,L),
   doReally(A), /*execute A in reality*/
     !,   /* commit to the result */
   online(ER,do(A,S),H,PolFut,UFut),
   Pol=(A : PolFut), U is R + UFut ;
   senseAction(A),
   doReally(A),  /* do sensing */
     !, /*commit to results of sensing*/
   online(ER,do(A,S),H,PolFut,UFut),
   Pol=(A : PolFut), U is R + UFut ;
   agentAction(A), stochastic(A,S,L),
   doReally(A), /*execute A in reality*/
     !,   /* commit to the result */
   senseEffect(A,S,SEff),
   diagnose(SEff,L,SN), /*What happened?*/
   online(ER,SN,H,PolFut,UFut),
   Pol=(A : PolF), U is R + UFut
  )
).
```

The on-line interpreter uses the Prolog cut (!) to prevent backtracking to the predicate $doReally$: we need this because once actions have been actually performed in the physical world, the robot cannot undo them.

In addition to predicates mentioned in section 2, the on-line interpreter uses the predicate *senseEffect*$(A, S1, S2)$, the predicate *diffSequence*(A, Seq) and the predicate $diagnose(S2, OutcomesList, S3)$. We describe below their meaning and show their implementation in Prolog.

Given the stochastic action A and the situation $S1$, the predicate *senseEffect*$(A, S1, S2)$ holds if $S2$ is the situation that results from doing a number of sensing actions necessary to differentiate between outcomes of the stochastic action A. The predicate *diffSequence*(A, Seq) holds if Seq is the sequence of sensing actions ($a_1; a_2; \ldots; a_n$) specified by the programmer in the domain problem representation: this sequence is differentiating if after doing all actions in the sequence the action chosen by 'nature' as the outcome of stochastic action A can be uniquely identified.

```
senseEffect(A,S,SE) :-  diffSequence(A,Seq),
    getSensorInput(S,Seq,SE).

getSensorInput(S,A,do(A,S)) :- senseAction(A),
  doReally(A). /*connect to sensors, get data
                for a free variable in A */

getSensorInput(S1,(A : Rest),SE) :-
   doReally(A),  /* connect to sensors,
                 get data         */
 getSensorInput(do(A,S1),Rest,SE).
```

The predicate $diagnose(S1, L, S2)$ takes as its first argument the situation resulting from getting a sensory input: it contains 'enough' information to disambiguate between different possible outcomes of the last stochastic action A. The second argument is the list L of all outcomes that nature may choose if the agent executes the stochastic action A in $S1$ and the third argument is the situation that is the result of nature's action which actually occurred. We can identify which action nature has chosen using the set of mutually exclusive test conditions *senseCond*(n_i, ϕ_i), where ϕ_i is a term representing a situation calculus formula: if ϕ_i holds in the current situation, then we know that nature has chosen the action n_i (n_i belongs to the list L).

```
diagnose(SE,[N],do(N,SE)) :-
        senseCond(N,C), holds(C,SE).

diagnose(SE,[N|Outcomes],SN):- senseCond(N,C),
       ( holds(C,SE), SN=do(N,SE) ;
         not holds(C,SE),
         diagnose(SE,Outcomes,SN) ).
```

Successful tests of the implementation described here were conducted in a real office environment on a mobile robot B21 manufactured by RWI. The low-level software was initially developed in the University of Bonn to control Rhino, another B21 robot, see [Burgard *et al.*, 1999] for details. The tests of implementation demonstrated that using the expressive set of Golog operators it is straightforward to encode domain knowledge as constraints on the given large MDP problem. The operator $optimize(p)$ proved to be useful in providing heuristics which allowed to compute sub-policies in real time. These preliminary tests have brought several new important issues, e.g., how the computation of a new policy off-line can proceed in parallel with executing actions from the policy on-line.

5 Discussion

The incremental Golog interpreter based on the single-step $Trans$-semantics is introduced in [De Giacomo and Levesque, 1999b]. The Golog programs considered there may include binary sensing actions. The interpreter considered in our paper is motivated by similar intuitions, but it is based on a different decision-theoretic semantics and employs more expressive representation of sensing. The paper [De Giacomo and Levesque, 1999a] introduces guarded sensed fluent axioms (GSFA) and guarded successor state axioms (GSSA) and assumes that there is a stream of sensor readings available at any time. These readings are represented by unary sensing functions (syntactically, they look like functional fluents). Because we introduce the representation for sensing actions, they can be mentioned explicitly in Golog programs or can be executed by the interpreter. The major advantage of our representation is that it does not require consistency of sensory readings with the action theory (this may prove useful in solving diagnostic tasks: [McIlraith, 1998]). The execution monitoring framework proposed in [De Giacomo *et al.*, 1998] assumes that the feedback from the environment is provided in terms of actions executed by other agents. Because in this paper we assume that the feedback is provided in terms of sensory readings, this may lead to development of a more realistic framework. An approach to integrating planning and execution in stochastic domains [Dearden and Boutilier, 1994] is an alternative to the approach proposed here.

6 Concluding remarks

Several important issues are not covered in this paper. One of them is monitoring and rescheduling of policies. Note that all actions in policies have time arguments which will be instantiated by moments of time: when the incremental interpreter computes an optimal policy, it also determines a schedule when actions have to be executed. But in realistic scenarios, when the robot is involved in ongoing processes extended over time, it may happen that a process will terminate earlier or later than it was expected.

The diagnostic task that the current version of the on-line interpreter solves is admittedly oversimplified. We expect that additional research integrating the on-line incremental interpreter with the approach proposed in [McIlraith, 1998] will allow us to formulate a more comprehensive version.

7 Acknowledgments

Thanks to Ray Reiter, Maurice Pagnucco, and anonymous reviewers for comments on preliminary versions of this paper. Sam Kaufman provided help with conducting tests on the mobile robot.

References

[Bacchus *et al.*, 1995] Fahiem Bacchus, Joseph Y. Halpern, and Hector J. Levesque. Reasoning about noisy sensors in the situation calculus. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1933–1940, Montreal, 1995.

[Boutilier *et al.*, 2000] C. Boutilier, R. Reiter, M. Soutchanski, and S. Thrun. Decision-theoretic, high-level robot programming in the situation calculus. In *Proc. of the 17th National Conference on Artificial Intelligence (AAAI'00)*, Austin, Texas, 2000.

[Burgard *et al.*, 1999] W. Burgard, A.B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. Experiences with an interactive museum tour-guide robot. *Artificial Intelligence*, 114(1-2), 1999.

[De Giacomo and Levesque, 1999a] G. De Giacomo and H. Levesque. Projection using regression and sensors. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, Stockholm, Sweden, 1999.

[De Giacomo and Levesque, 1999b] G. De Giacomo and H.J. Levesque. An incremental interpreter for high-level programs with sensing. In Levesque and Pirri, editors, *Logical Foundations for Cognitive Agents: Contributions in Honor of Ray Reiter*, pages 86–102. Springer, 1999.

[De Giacomo *et al.*, 1998] G. De Giacomo, R. Reiter, and M.E. Soutchanski. Execution monitoring of high-level robot programs. In *Principles of Knowledge Representation and Reasoning: Proc. of the 6th International Conference (KR'98)*, pages 453–464, Trento, Italy, 1998.

[Dearden and Boutilier, 1994] Richard Dearden and Craig Boutilier. Integrating planning and execution in stochastic domains. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 162–169, 1994.

[Funge, 1998] J. Funge. *Making Them Behave: Cognitive Models for Computer Animation, Ph.D. Thesis*. Dept. of Computer Science, Univ. of Toronto, 1998.

[Grosskreutz, 2000] H. Grosskreutz. Probabilistic projection and belief update in the pgolog framework. In *The 2nd International Cognitive Robotics Workshop, 14th European Conference on AI*, pages 34–41, Berlin, Germany, 2000.

[Lakemeyer, 1999] G. Lakemeyer. On sensing and off-line interpreting in golog. In Levesque and Pirri, editors, *Logical Foundations for Cognitive Agents: Contributions in Honor of Ray Reiter*, pages 173–189. Springer, 1999.

[Levesque, 1996] H.J. Levesque. What is planning in the presence of sensing? In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, volume 2, pages 1139–1145, Portland, Oregon, 1996.

[McIlraith, 1998] S. McIlraith. Explanatory diagnosis: Conjecturing actions to explain obsevations. In *Principles of Knowledge Representation and Reasoning: Proc. of the 6th International Conference (KR'98)*, pages 167–177, Italy, 1998.

[Pirri and Finzi, 1999] F. Pirri and A. Finzi. An approach to perception in theory of actions: part 1. *Linköping Electronic Articles in Computer and Information Science*, 4(41), 1999.

[Pirri and Reiter, 1999] F. Pirri and R. Reiter. Some contributions to the metatheory of the situation calculus. *Journal of the ACM*, 46(3):261–325, 1999.

[Reiter, 2000] R. Reiter. *Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems*. http://www.cs.toronto.edu/~ cogrobo/, 2000.

[Scherl and Levesque, 1993] R. Scherl and H.J. Levesque. The frame problem and knowledge producing actions. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 689–695, Washington, DC, 1993.

[Soutchanski, 2001] M Soutchanski. A correspondence between two different solutions to the projection task with sensing. In *The 5th Symposium on Logical Formalizations of Commonsense Reasoning*, New York, USA, 2001.