Planning with Trial and Errors

Rajdeep Niyogi
Department of Electronics and Computer Engineering
Indian Institute of Technology Roorkee 247667, India
rajdpfec@iitr.ernet.in

***Abstract*—Agent-based systems have gained immense popularity in the last decade or so. Reasoning methods such as planning and decision-making under uncertainty applied to agents and multi-agent systems is an important area of research in agent-based systems. Most works in planning in uncertain environments incorporate sensing actions. We wish to address planning problems when there are no such sensing actions. For this, we introduce the notion of 'try actions' and develop a logic PDL_try that is an extension to propositional dynamic logic (PDL). The logic also allows specification of the implicit knowledge of an agent. We illustrate planning in our logic with a new adaptation of the popular blocks world domain. Example programs (plans) are discussed in detail for single agent and two agent situations.**

Keywords: Deductive planning, PDL, try actions

I. Introduction

One of the primary goals of Artificial Intelligence is to design intelligent agents. Some typical tasks of intelligent agents are searching, planning, and learning. Most of the planning research in classical planning is about finding a sequence of actions to achieve some goal. Deductive planning [10], [8], on the other hand, considers plans as programs and planning as kind of program synthesis. In classical planning, plan synthesis process is algorithmic, i.e., to design an efficient algorithm (planner) that finds a plan. In deductive planning, plan synthesis is reduced to theorem-proving [10], i.e., proving a theorem implies finding a plan. The difficulty in classical planning is due to the large size of the state-space. In deductive planning, the difficulty is in reasoning about plans, which implies reasoning about correctness, termination conditions of programs.

Uncertainty in the real-world usually manifests in the following ways: (i) incomplete state information, and (ii) nondeterministic outcomes of actions. It has been shown in [10], [8], [1] that even for simple domains involving uncertainty, there exist no classical plans. In such scenarios it is meaningful to specify plans as programs. In this paper we consider planning under uncertainty where the world manifests the above features.

The idea is to specify and reason about plans viewed as programs. We wish to look at logical formalisms for reasoning about actions of an agent. Propositional dynamic logic (PDL) [7] is a natural vehicle for studying how actions (plans) change truth values. There has been some work in automated planning [11], [6], [12], and agent programming [13] that employs PDL. Our work also employs and enhances PDL. However, the directions in which PDL is extended differs from others [11], [6], [12]. A detailed discussion in this regard is given in Section V.

The rest of the paper is structured as: in Section II we give some motivations, in Section III we present the logic PDL_try, in Section IV we give detailed example programs for single agent and multiple agent scenario, in Section V we discuss on the related works, and in Section VI we conclude.

II. Motivation

One way to overcome uncertainty in unpredictable environments is by using sensing actions. A sensing action is like an observer that does not change the state of the world. Some examples of sensing actions include litmus test, a read action, a look/observe action, and the Unix commands 'ls,pwd'. Sensing actions are considered in [8], [1], [4], [5], [9]. However, Spalazzi and Traverso [12] consider planning in a reactive system where sensing actions may change the state of the world and moreover such actions may fail (actions may abort). Thus, when sensing actions fail, the plans–programs–involving sensing actions cannot be executed.

We wish to address the following: *how can we do planning in (some) uncertain environments when there are no sensing actions similar to those mentioned above.*

The example domain that we consider is adapted from the popular Blocks World (BW) domain. The fluents in BW domain are $ontable(x), clear(x)$, $handempty$, $on(x, y)$. The operators are $pickup(x), putdown(x)$, $stack(x, y), unstack(x, y)$. The operator $stack(x, y)$ is used to denote put block x on top of block y; $unstack(x, y)$ is the reverse of $stack(x, y)$. An action is a ground instance of an operator. For example, if we have 3 blocks-A,B,C, $pickup(A)$, $putdown(C), stack(A, B)$, etc., are the actions. Let the initial configuration be that all the blocks are placed on the table. The final configuration is a tower of blocks with A at the bottom, C at the top, and B on top of A.

The planning problem is to find a plan that transforms the initial configuration to the final configuration.

Modified BW domain: We assume that the blocks are of different weights. In the original BW domain, the preconditions of $pickup(x)$ are $clear(x)$, $handempty$, and $ontable(x)$. We redefine $pickup(x)$ by attaching one more pre-condition: block is not heavy. In the modified domain, we assume that the planning agent has no information regarding the weight of a block. We define a new operator $apply\_lever(x, y)$ that places a block x on top of block y, when both the blocks

978-1-4244-4711-4/09/$25.00 ©2009 IEEE

x, y are heavy, and there is nothing on top of x, y. During planning a heavy block should not rest on a light block. Consider an example scenario given in figure 1. Let the initial configuration be that all the blocks are placed on the table. The final configuration is a tower of blocks such that the lighter blocks rest on top of heavy blocks.

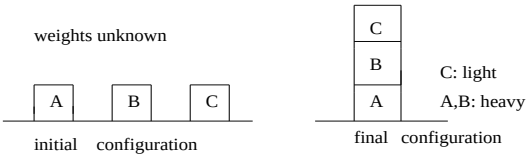

Fig. 1. Modified Blocks World Domain

The modified BW (MBW) domain differs from the BW domain in the following:

1. In BW domain, the initial state is completely known. States are completely observable. In MBW domain, states are partially observable.

2. In BW domain, the action outcomes are deterministic. In MBW, only the 'try' actions–that we introduce here–are nondeterministic.

Some Possible approaches to the Modified BW domain

1. Using sensing actions that provide information regarding weights of the blocks. Let us assume that these actions act as observers and that they do not fail. Then, there exists a plan (program), similar to those given in [8], [1], that solves the planning problem. It may be noted that *there is no sequential plan* that solves the above problem.

2. Using sensing actions that may fail (abort). Then, there exists a plan (program) specified in the logic FSP [12], an extension to propositional dynamic logic, for the above problem.

3. [Our approach] We assume that there are no sensing actions. The planning agent has no information about the weight of a block. But it can always 'try' some action to find whether a block is heavy or not. We introduce the operator $try\_pickup(x)$ that is quite similar to $pickup(x)$, except that unlike $pickup(x)$ now $try\_pickup(x)$ may be enabled at the initial state. If it succeeds, its outcome is exactly that of $pickup(x)$, i.e., $holding(x)$ becomes true. Now suppose that $try\_pickup(x)$ fails. (In this paper we assume that the notion of failure is with respect to goal attainment and not that of failure of actions (abort) as in [12].) Then the agent comes to know that the block is heavy, implying that $pickup(x)$ cannot be executed at that state. So the try action allows the agent to update its knowledge about the world. A try action is nondeterministic with two possible outcomes depending on whether it succeeds or it fails.

In order to specify plans (programs), we propose a logic that is an extension to propositional dynamic logic (PDL) in two directions–(i) it incorporates such try actions, and (ii) it allows specification of the implicit knowledge of an agent.

III. A Propositional Dynamic Logic with Try Actions

In propositional dynamic logic (PDL) [7], complex formulas and programs are built from atomic propositions and primitive actions. Our logic PDL_try is an enhancement of PDL where we introduce try actions and a knowledge operator.

Our logic consists of two sets of symbols: P–the set of atomic propositions and A–the set of primitive actions. We include in A the set of 'try actions', i.e., try_a is a primitive action corresponding to an action $a \in A$. If try_a is an action for some $a \in A$, we include the propositions $\sigma(try\_a)$ and $\tau(try\_a)$ in the set of atomic propositions P to indicate that the 'try action' has succeeded or failed respectively. By success we mean that an action has achieved the desired postcondition. By failure we mean that an action did not achieve the desired postcondition.

Syntax: From P and the set of actions (including the 'try actions') the set ϕ of propositions and the set π of plans is defined by mutual induction as:

$$\phi := p \in P \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid [\pi]\phi_1 \mid \langle\pi\rangle\phi_1 \mid K\phi$$
$$\pi := a, try\_a \in A \mid \pi_1; \pi_2 \mid \pi_1 \cup \pi_2 \mid \pi_1^* \mid \phi?$$

Semantics: The semantics of PDL is defined in terms of a labeled transition system (LTS). Since PDL_try is similar to PDL, so it will also be defined in terms of an LTS. The difference in the two logics is due to the presence of the constructs try and K that denote 'try actions' and 'knowledge' respectively.

Thus we define the transition system (or frame) as $F = (S, \sim, \rightarrow)$, where

- S is a (finite) set of possible-world states,
- $\sim$ is the equivalence relation called the indistinguishability relation of the agent; $\sim$ defines an equivalence class,
- $\rightarrow \subseteq S \times A' \times S$ is the transition relation, where $A' = A \cup \{try\_a \mid a \in A\}$, such that,
  - $\rightarrow_a \subseteq S \times S$, $a \in A$, and
  - $\rightarrow_{try\_a} \subseteq S \times S$, $a \in A$

and the following conditions(1-4), given below, hold in F.

The information state of the agent is captured by the equivalence class. The agent does not know the true state of the world. The agent views the actual (true) state of the world as a set of states that it thinks is possible. For example, if the weight of a block in the BW domain is not known, then the agent considers two states possible–one in which $block\_heavy(x)$ and the other in which $\neg block\_heavy(x)$. So to the agent a set of states is possible when it is in some state. try_a is a way by which the agent can learn and refine the partition.

For the transition system F, we define the following conditions as:

- **Condition1**. $\forall(s_1, s_2, a)$ if $s_1 \sim s_2$ then try_a is enabled at s_1 iff try_a is enabled at s_2
- **Condition2**. $\forall(s_1, s_2, s_2', a)$ if (s_1, a, s_2) and (s_1, try_a, s_2') then $s_2 = s_2'$
- **Condition3**. $\forall(s, a)$ if try_a is enabled at s, then $\exists s'$ s.t. $s \sim s'$ and a is not enabled at s'
- **Condition4**. $\forall(s_1, s_2, s_1', s_2', a)$ if $s_1 \sim s_2$ and (s_1, a, s_1') and (s_2, a, s_2'), then $s_1' \sim s_2'$

Condition1 says that try_a is enabled in all the equivalent states. We can express this condition in our logic as: $\langle try\_a\rangle\top \supset K\langle try\_a\rangle\top$, the formula is to be read as: if a 'try' action is enabled at a state then the agent knows it.

Condition2 says that if the state resulting from performing an action a at s_1 is s_2 and that of performing try_a at s_1 is s_2', then s_2' is no different from s_2.

Condition3 says that in any set of equivalent states, there always exist a state where the basic action corresponding to the 'try' action is not enabled. We can express this condition in our logic as: $\langle try\_a\rangle\top \supset \neg K\langle a\rangle\top$, the formula is to be read as: if a try_a is enabled at a state s then it implies that the agent does not know whether a is enabled at s. This is obvious, since if the agent knew that a is enabled then why would it try.

Condition4 says that if there is no information gain in a set of equivalent states, then the resulting states are also equivalent. This condition is equivalent to saying in our logic: $[a]K\alpha \equiv K[a]\alpha$

A model is defined as $M = (F, V)$ where $F = (S, \sim, \rightarrow)$ and V is a valuation function $V : S \rightarrow 2^P$ such that,

–if (s, a, s') and (s, try_a, s') then $\sigma(try\_a) \in V(s')$ (by this we mean that try_a succeeded at s)

–if (s, try_a, s') and a is not enabled at s then $\tau(try\_a) \in V(s')$ (by this we mean that try_a failed at s)

Given a model M, a state in S, the satisfaction relation $\models$ for different types of formulas α is given as:

$M, s \models p$ iff $p \in V(s)$ for $p \in P$
$M, s \models \alpha_1 \vee \alpha_2$ iff $M, s \models \alpha_1$ or $M, s \models \alpha_2$
$M, s \models \neg\alpha$ iff $M, s \not\models \alpha$
$M, s \models [\pi]\alpha$ iff for all t, $s \rightarrow_\pi t$ and $M, t \models \alpha$
$M, s \models \langle\pi\rangle\alpha$ iff there exists t, $s \rightarrow_\pi t$ and $M, t \models \alpha$
$M, s \models K\alpha$ iff for all s', $s' \sim s$ and $M, s' \models \alpha$

The transition relation for any general plan π is obtained recursively from the transition relation of the primitive actions. This is done in a standard way as in PDL.

$s \rightarrow_{\pi_1;\pi_2} t$ iff there is a u such that $s \rightarrow_{\pi_1} u$ and $u \rightarrow_{\pi_2} t$
$s \rightarrow_{\pi_1\cup\pi_2} t$ iff $s \rightarrow_{\pi_1} t$ or $s \rightarrow_{\pi_2} t$
$s \rightarrow_{\pi_1^*} t$ iff there are $s_1, \ldots, s_n$ such that $s = s_1 \rightarrow_{\pi_1} s_2 \rightarrow_{\pi_1} \ldots \rightarrow_{\pi_1} s_n = t$
$s \rightarrow_{\phi?} s'$ iff $s' = s$ and $s' \models \phi$

IV. Illustrations

We shall illustrate with the MBW domain the plans (programs) for the logic PDL_try. Let us assume that there are 3 blocks (A,B,C). See figure 1. Let the initial configuration (I) be that all the blocks are placed on the table. The final configuration (G) is a tower of blocks such that the lighter blocks rest on top of heavy blocks. The weights of the blocks are not known initially. The planning problem is to find a plan (program) π that transforms I to G. That is, find π such that $I \supset [\pi]G$, or (equivalently expressed as a Hoare-tuple) $\{I\}\pi\{G\}$ holds.

A. Single Agent

We give below the program π, that solves the given problem, when $try\_pickup(A)$ is chosen as the first action. The programs are similar when the first action chosen is either $try\_pickup(B)$ or $try\_pickup(C)$. The program $if\ \varphi\ then\ \alpha\ else\ \beta \equiv \varphi?;\alpha \cup \neg\varphi?;\beta$. The changes in agent's knowledge is also given (within comments).

$\pi = try\_pickup(A);$
$if\ \sigma(try\_pickup(A))\ then\ (\pi_1 \cup \pi_2)$ // $K\ \neg heavy(A)$
$else\ try\_pickup(B);$ // $K\ heavy(A)$
$if\ \sigma(try\_pickup(B))\ then\ \pi_3$ // $K\ \neg heavy(B)$
$else\ try\_pickup(C);$ // $K\ (heavy(A) \wedge heavy(B))$
$if\ \sigma(try\_pickup(C))\ then\ \pi_4$ // $K\ \neg heavy(C)$
$else\ apply\_lever(C, B); apply\_lever(A, B)$ // $K\ (heavy(A) \wedge heavy(B) \wedge heavy(C))$

$\pi_1 = putdown(A);$
$try\_pickup(B);$
$if\ \sigma(try\_pickup(B))\ then\ P_1$ // $K\ \neg heavy(B)$
$else\ try\_pickup(C);$ // $K\ heavy(B)$
$if\ \sigma(try\_pickup(C))\ then\ P_2$ // $K\ \neg heavy(C)$
$else\ P_3$ // $K\ (\neg heavy(A) \wedge heavy(B) \wedge heavy(C))$

$\pi_2 = putdown(A);$
$try\_pickup(C);$
$if\ \sigma(try\_pickup(C))\ then\ P_2$ // $K\ \neg heavy(C)$
$else\ try\_pickup(B);$
$if\ \sigma(try\_pickup(B))\ then\ P_1$ // $K\ \neg heavy(B)$
$else\ P_3$ // $K\ (\neg heavy(A) \wedge heavy(B) \wedge heavy(C))$

$\pi_3 = putdown(B);$
$try\_pickup(C);$
$if\ \sigma(try\_pickup(C))\ then\ P_4$ // $K\ \neg heavy(C)$
$else\ P_5$ // $K\ (heavy(A) \wedge \neg heavy(B) \wedge heavy(C))$

$\pi_4 = putdown(C); apply\_lever(A, B); pickup(C); stack(C, A)$ // $K\ (heavy(A) \wedge heavy(B) \wedge \neg heavy(C))$

$P_1 = stack(B, C); pickup(A); stack(A, B)$ // A,B light; C anything

$P_2 = stack(C,B); pickup(A); stack(A,C)$ // A,C light; B anything
$P_3 = apply\_lever(C,B); pickup(A); stack(A,B)$ // B,C heavy; A light
$P_4 = stack(C,A); pickup(B); stack(B,C)$ // A heavy; B,C light
$P_5 = apply\_lever(A,C); pickup(B); stack(B,A)$ // A,C heavy; B light

The program (π), above, solves the planning problem $I \supset [\pi]G$.

B. Two Agents

There are two agents collaborating with each other to solve the above problem in the MBW domain. Let $try\_pickup(A)^1, try\_pickup(B)^2$ be the first actions chosen by the agents 1 and 2 respectively. Before the actions were performed there are 4 world states that the agents collectively considers possible for the two blocks. After the actions are performed each agent knows the correct world state. Now the agents communicate among themselves to know the outcomes of the individual actions. As a result of the communication, the knowledge of the agent changes, thereby reducing its uncertainty of the world. The if-fi construct is defined as $if\ \varphi_1 \supset \alpha_1 \mid \ldots \mid \varphi_n \supset \alpha_n fi \equiv \varphi_1?;\alpha_1 \cup \ldots \cup \varphi_n?;\alpha_n$ is the alternative guarded command. Thus the overall program π will look like:

$\pi = try\_pickup(A)^1 \mid\mid try\_pickup(B)^2$; // $\mid\mid$ denotes actions performed concurrently
// Now the agents communicate. There are 4 possible outcomes of the 'try' actions. Consider the case when both the try actions succeed. Thus, $K_1\ \neg heavy(A)$ and $K_2\ \neg heavy(B)$–so agent 2 performs $stack(B,C)$. After communication, $K_2\ \neg heavy(A)$ and $K_1\ \neg heavy(B)$–so agent 1 performs $stack(A,B)$. Moreover the mutual knowledge of the agents become $K_1K_2\ \neg heavy(A)$ and $K_2K_1\ \neg heavy(B)$.

$if\ (\sigma(try\_pickup(A)^1) \wedge \sigma(try\_pickup(B)^2)) \supset stack(B,C)^2; stack(A,B)^1$
$\mid (\sigma(try\_pickup(A)^1) \wedge \tau(try\_pickup(B)^2)) \supset P_1$
$\mid (\tau(try\_pickup(A)^1) \wedge \sigma(try\_pickup(B)^2)) \supset P_2$
$\mid (\tau(try\_pickup(A)^1) \wedge \tau(try\_pickup(B)^2)) \supset P_3$
fi // end of program π

P_1 is given below. P_2 is similar.
$P_1 = try\_pickup(C)^2;$
$if\ \sigma(try\_pickup(C)^2)\ then\ stack(C,B)^2; stack(A,C)^1$
$else\ apply\_lever(C,B)^2; stack(A,C)^1$
$P_3 = apply\_lever(A,B)^1; try\_pickup(C)^1;$
$if\ \sigma(try\_pickup(C)^1)\ then\ stack(C,A)^1$
$else\ apply\_lever(C,A)^1$

V. Related Work

We compare our work with that of others' based on the following broad areas.

Sensing actions have been addressed in [4], [5], [1], [9], [12]. In [4], [5] action description languages are proposed. It allows specification of conditional plans arising out of the outcomes of sensing actions. In [9] plans are viewed as programs and a theory based on situation calculus is developed for reasoning about actions. A theory of sensing actions based on a transition function approach is given in [1]. In all these papers except [12] sensing actions are like observers–their outcomes do not change the world state. We consider 'try actions' that change the world state; so it differs from the sensing actions in [4], [5], [1], [9].

World state and information state. In uncertain environments, we need to distinguish between the actual state of the world and the information state of an agent (agent's knowledge of the world). The paper [1] considers three types of states–world state, knowledge state (k-state), and combined state (c-state) that consists of a world state and a k-state. Transition functions map a pair of c-state and an action to another c-state. In our work, an agent's information state is modeled by the equivalence relation $\sim$ on the possible-world states.

Failure of actions. In [12] some example scenarios from reactive planning domains are considered where sensing actions fail. Here success/failure of actions are not related to goal attainment/non-attainment. Actions fail when they abort. In our work success/failure of actions are related to goal attainment/non-attainment. Another line of work, though not closely related to ours, is that in [3], [2]. They consider a robotic application where plans encoding iterative trial-and-error strategies like 'pick up a block until succeed' are the only acceptable solutions [3]. The failures of actions in [3], [2] are similar to those in [12], so the iterative plans in the former can be specified using FSP [12].

Knowledge operator. Explicit use of knowledge operator when sensing actions are present is considered in [5], [9], [6]. Formulations of knowledge based on situation calculus is given in [5], [9]. They define the operator $Know(\phi, s)$ to mean that the agent knows ϕ at situation s, which is true when ϕ holds at all situations that the agent considers possible at s. In [6], the meaning of $Know(\phi)$ is based on the possible-worlds semantics, where ϕ is restricted to atomic propositions. The definition of knowledge operator $K\phi$, given in this paper, is also based on possible-world semantics, but ϕ may involve complex plans as well.

Dynamic logic based planning. A logic for failure, sensing, and planning (FSP) is proposed in [12]. FSP is an extension of PDL with the converse operator. A loop-free fragment of

PDL is considered in [11], [6] that concerns programs with if-then-else construct. In [6] the $Know(\phi)$ operator occurs in programs, when sensing actions are used. PDL_try differs from FSP in that (i) converse operators are not used in the former, (ii) the semantics of failure/success are different, (iii) the 'try actions' and the knowledge operator $K\phi$ are not used in the latter. PDL_try uses the full expressive power of PDL, which is not the case in [11], [6].

VI. Conclusions and Future Work

The paper made a first attempt to deal with planning under uncertainty when there are no conventional sensing actions. We demonstrated the usefulness of 'try actions' with the MBW domain taking both single agent and multi agent cases. As part of our ongoing/future work we wish to address: (i) to establish the complexity of the plan existence problem for PDL_try, (ii) to show how plan extraction can be done for PDL_try formulas, and (iii) to extend PDL_try for multi agent systems, and then to address the problems given in (i) and (ii).

Acknowledgement

We are grateful to R. Ramanujam for the helpful contributions. Part of the work was done while the author was visiting IMSc, Chennai. We thank the anonymous referees for their valuable comments. This research is supported in part by a faculty initiation grant (scheme A) from SRIC, IIT Roorkee.

References

[1] C. Baral and T. Son, "Formalizing sensing actions–a transition function based approach," *Artificial Intelligence*, vol. 125(1-2), pp. 19–91, 2001.

[2] A. Cimatti, M. Pistore, M. Roveri, and P. Traverso, "Weak, strong, and strong cyclic planning via symbolic model checking," *Artificial Intelligence*, vol. 147(1-2), pp. 35–84, 2003.

[3] M. Daniele, P. Traverso, and M. Vardi, "Strong cyclic planning revisited," in *Proceedings of the 5th European Conference on Planning*. Springer-verlag, 1999, pp. 35–48.

[4] O. Etzioni, S. Hanks, D. Weld, D. Draper, N. Lesh, and M. Williamson, "An approach to planning with incomplete information," in *Proceedings of Knowledge Representation*, 1992, pp. 115–125.

[5] K. Golden and D. Weld, "Representing sensing actions: The middle ground revisited," in *Proceedings of Knowledge Representation*, 1996, pp. 213–224.

[6] R. Golden and M. Boddy, "Expressive planning and explicit knowledge," in *Proceedings of the Third Artificial Intelligence Planning Systems*, 1996, pp. 110–117.

[7] D. Harel, D. Kozen, and J. Tiuryn, *Dynamic Logic*. MIT Press, 2000.

[8] H. Levesque, "Planning with loops," in *Proceedings of IJCAI*, 2005, pp. 311–316.

[9] ——, "What is planning in the presence of sensing?" in *Proceedings of AAAI*, 1996, pp. 1139–1146.

[10] Z. Manna and R. Waldinger, "How to clear a block: A theory of plans," *Journal of Automated Reasoning*, vol. 3, pp. 343–377, 1987.

[11] S. Rosenschien, "Plan synthesis: a logical perspective," in *Proceedings of IJCAI*, 1981, pp. 331–337.

[12] L. Spalazzi and P. Traverso, "A dynamic logic for acting, sensing, and planning," *Logic Computation*, vol. 10(6), pp. 727–821, 2000.

[13] M. van Riemsdijk, F.S.deBoer, and J. C. Meyer, "Dynamic logic for plan revision in intelligent agents," in *Proceedings of the 5th International Workshop on Computational Logic in Multi agent systems*, 2005, pp. 16–32.