PLANNING ACTIVITIES IN SPACE

Kai-Hsiung Chang

Department of Computer Science and Engineering
Auburn University, Auburn, Alabama 36849-5347

0. ABSTRACT

This paper presents three aspects of planning activities in space. These include, (1) generating plans efficiently, (2) coordinating actions among multiple agents, and (3) recovering from plan execution errors. Each aspect will be discussed separately.

1. INTRODUCTION

An autonomous space station is required to formulate its own action plan after receiving a mission command. In order to accomplish this goal, a system that is able to generate action plans for various agents, coordinate actions among agents, and decide on recovery plans for execution errors will be required. This paper describes some research works of these areas. The author assumes that the reader already has a basic knowledge on planning.

2. A TWO-PHASE PLANNING STRATEGY

This approach would generate plans efficiently according to the goal requirements. In this strategy, a planning process is divided into two phases, goal analysis and plan generation. The idea of goal analysis is to reduce the fruitless search space at the start of the planning process and to provide a correct outline for the generation of plans. In the block stacking example of Figure 1, most human experts know that in order to build a structure, the lower part has to be built first and the lowest block has to be put on the table. With a simple analysis using these two heuristic rules, an expert can conclude quickly that (ON B C) should be achieved before (ON A B) and that C should stay on the table. If a planning system also adopts this heuristic analysis, the same conclusion can also be reached. We believe that this analysis is close to a human planning model and is more efficient for solving a problem.

In the plan generation phase, a goal-oriented hierarchical operator representation technique is used to avoid the time consuming operator searching process. Usually, a goal can be achieved by several different operators; but some of them may not be applicable at a specific instance, and some of them may have side effects that would cause problems later. Trail-and-error search process is used to select operators in most conventional systems. This time-consuming search process can be avoided. First, if each operator is named by the goal it would achieve, there is no need to search for operator candidates. Secondly, within each operator representation, a sequence of detailed

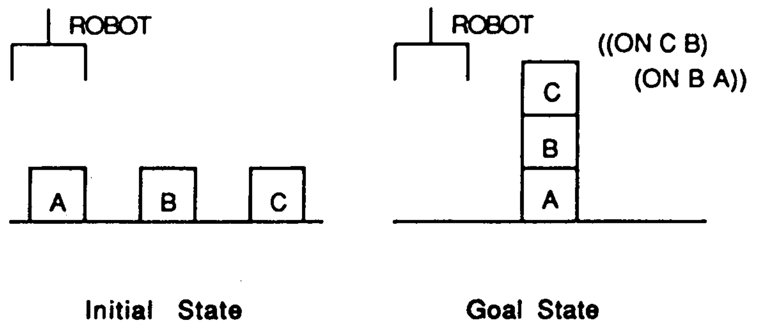


Figure 1 A simple block stacking problem.

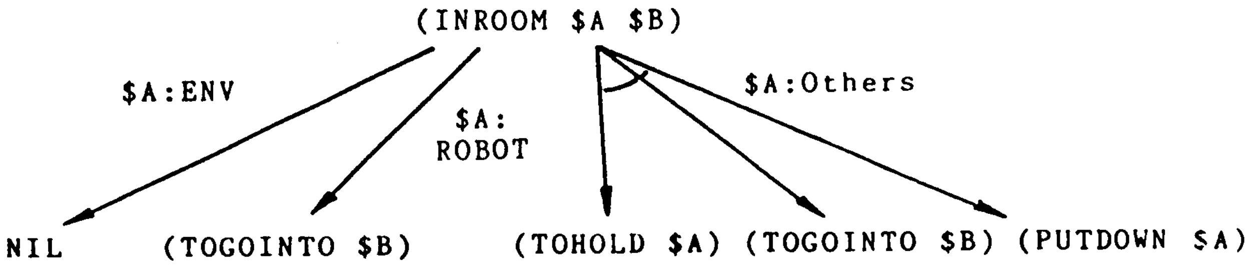


Figure 2 Operator hierarchy of (INROOM $A $B)

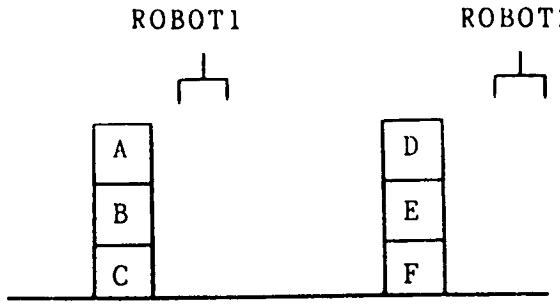
operators can be selected to satisfy special requirements of different situations. A sample operator hierarchy is shown in Figure 2. In this hierarchy, the goal is to move object $A into room $B. Since the name and the goal of the operator are identical, this representation is considered goal-oriented. In this example (INROOM $A $B) must be refined into a sequence of detailed operators before its execution. The sequence selection is determined by the requirements of the goal and the world state. Here, if $A is an ENV (environmental) object, like DOOR, which can not be moved, then nothing has to be done. If $A is a ROBOT, which can move, then the ROBOT just has (TO-GO-INTO $B). Finally, if $A is something else, then the ROBOT has (TO-HOLD $A), then (TO-GO-INTO $B), and then (PUT-DOWN $A). An abstract operator, like (INROOM $A $B), can be refined into more detailed operators by simple condition matching. Detailed examples and applications of this approach can be found in [1,2].

3. A MULTIAGENT PLANNING SYSTEM

In a multiagent environment, the resource sharing and action coordination must be managed carefully. This is critical to the success of an integrated system which involves multiple agents[3]. In our approach, three features have been proposed.

They are meta-level planning, agent-oriented dynamic task assignment, and breakable and unbreakable action sequences.

- Meta-level Planning: The purpose is to transform an original goal (problem) statement into a plan outline that is easier to pursue. The transformation includes grouping and ordering original goal components, adding new goal elements, and posting constraints. A typical example is shown in Figure 3. In the first step, the system groups subgoals according to resource. In the second step, it uses domain knowledge to determine the subgoal pursuing sequence in each group.



GOAL STATEMENT:

```
((ON B C) (ON A B)
 (ON D E) (ON C TABLE)
 (ON E F) (ON F TABLE))
```

Figure 3 A typical multiagent planning problem.

STEP1: Parallel Groups of Subgoals

```
(= ((ON B C) (ON A B) (ON C TABLE))
   ((ON D E) (ON E F) (ON F TABLE)))
```

STEP2: Ordered Subgoals and Groups

```
(= ((ON C TABLE) (ON B C) (ON A B))
   ((ON F TABLE) (ON E F) (ON D E)))
```

The equal sign ("=") of STEP1 shows that there are two parallel groups of subgoals, which can be pursued by different agents. The result of STEP2 shows the sequence of pursuing subgoals within a group. For example, the sequence of stacking the first block pile is (ON C TABLE), (ON B C), (ON A B).

- Agent-oriented dynamic task assignment: This is to find out what an agent can do at different times. The planner always tries to assign one or a team of free agents to be in charge of one group of related subgoals. The assignment is determined by the features and the status of agents, the requirements of the task, and the constraints posted during the meta-level planning. Actions of each agent are then generated accordingly. Normally, an agent works for its own subgoal groups. However, exceptional condition is allowed for an agent to do unanticipated tasks.

- _Breakable_ _and_ _unbreakable_ _action_ _sequences_: The idea is to distinguish the unbreakable actions that must be executed by the same agent from those breakable actions that can be executed by different agents. This provides (a) cooperative tasks between agents can be identified without difficulty, (b) agent utility can be improved, and most importantly, (c) cumbersome reasoning for concurrent actions is eliminated. Detailed report will be published in the near future.

4. EXECUTION ERROR RECOVERY

Normally, a plan must be carried out in a world whose behavior cannot be predicted exactly, so one must be prepared for failures during execution[5]. A system that is capable of handling such failures is presented. One point this system has made is that it modifies only those parts of a plan that is absolutely necessary. The planning process involves the hierarchical expansion of abstract goals (or actions) into detailed actions. This in essence, generates a tree structure (called expansion tree or plan tree) with the leaves as the primitive actions that constitute the final plan. In order to aid in the error recovery process, a second tree called the decision tree is used. This is similar to the one proposed in [4]. The nodes in the decision tree are in one-to-one correspondence with the decisions made during the construction of that plan. Each node in this tree has a two way pointer from it to the nodes in the plan tree, which was created as a direct consequence of its decisions. The error recovery process consists of error identification, classification, and recovery.

4.1 ERROR IDENTIFICATION

Two methods have been used to identify errors in the plan execution monitoring. They are condition-oriented and object oriented approaches.

CONDITION-ORIENTED APPROACH

Since the problem is to identify errors, one must look for violations of conditions that need to be true at different parts of a plan. Several conditions are considered. 1. Preconditions. They are predicates that must be true before an action can be executed. 2. Expansion conditions. These are the status of world on which the expansion of a node depends. 3. Decision Conditions. These are the decisions made during the planning process and are based on a predefined heuristic function.

With these condition classifications, the identification of errors can be accomplished by comparing the current world state to the conditions recorded in the decision tree.

OBJECT-ORIENTED APPROACH

All the objects involved in the domain can be classified as critical or non-critical. Normally, a non-critical object is either an environmental object on which none or limited actions can be performed, or an agent that can perform actions; a critical object is one on which actions can be performed. If an error involves a non-critical object, only local readjustment needs to be made. If the error involves a critical objects, but the predicate involved does not fall into the critical category (door locked is a critical predicate), then local readjustment is needed but changes need to be propagated. This is similar to the violation of precondition case. However, if the predicate involved is critical, as the locked door, then a major replanning is needed.

4.2 ERROR CLASSIFICATION AND RECOVERY

Before an error can be cleared from the "world", the system has to recognize the type of the error so that an appropriate modification can be taken. Four error categories are used in our system.

1. Non-critical error: The modification consists of going one level higher in the plan hierarchy and adding a subplan when an assumed condition has failed, or removing a subplan when the goal was already achieved. This will not affect the rest of the plan in any way. Most expansion condition violations fall under this category.
2. Major error, but not critical: It is handled just like category 1. But the rest of the plan might be affected. So any changes should be propagated along. Precondition and some expansion condition violations belong to this category.
3. Critical error: This requires the abandoning and the replanning of certain subplans. Any changes should be propagated. Decision condition violations fall under this category.
4. Unrecoverable error: No modification takes place and the execution is aborted. A typical example in the blocks world is the malfunction of the robot arm, which will prevent any further actions. Human operators must be informed to resolve the error.

5. REFERENCES

[1] Chang, K.H. and Wee, W.G.,"Planning with analysis", Proc. 2nd IEEE Conf. on Artificial Intelligence Applications, 1985, pp. 275-280

[2] Chang, K.H. and Wee, W.G., "A Knowledge-based Planning System for Mechanical Assembly Using Robots", IEEE Expert, 1988. (To appear)

[3] Georgeff, M.P. "The Representation of Events in Multiagent Domain", Proceedings AAAI-86, 1986, pp. 70-76.

[4] Hayes, P. J. "A representation for robot plans". Proc. IJCAI, 1975, Tbilisi, USSR, pp. 181-188

[5] Wilkins, D. E. "Recovering from execution errors in SIPE". Computational Intelligence I, 1985, pp. 33-45