HIERARCHICAL PLANNING: DEFINITION AND IMPLEMENTATION

Technical Note 370

December 20, 1985

By: David E. Wilkins, Computer Scientist
Representation and Reasoning Group

Artificial Intelligence Center
Computer Science and Technology Division

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

The research reported herein is supported by the Air Force Office of Scientific Research, Contract No F49620-79-C-0188, SRI Project 7898.

Abstract

There is considerable ambiguity involved in hierarchical planning. We present a definition of the latter, and examine several of the reasons for this confusion. An explication of hierarchical-planning implementations entails two distinct notions: *abstraction level* and *planning level*. A problem in currently implemented planners that is caused by mixing these two levels is presented and various remedies suggested. Two solutions that have been implemented in the current SIPE planning system are described.

Category: Problem Solving

Keywords: planning, hierarchical planning, granularity, abstraction, goal, operator

1 Definition

It is generally recognized that planning in realistic domains requires planning at different levels of abstraction [3]. This allows the planner to manipulate a simpler, but computationally tractable, theory of its world. The combinatorics of concatenating the most detailed possible descriptions of actions would be overwhelming without the use of more abstract concepts. This has resulted in numerous hierarchical-planning systems. However, hierarchical levels and hierarchical planning mean quite different things in different planning systems, as is explained in the next section.

In our view, the essence of hierarchical planning (and a necessary defining condition) is the use of different levels of abstraction both in the planning process and in the description of the domain. An *abstraction level* is distinguished by the granularity [3], or fineness of detail, of the discriminations it makes in the world. From a somewhat more formal standpoint, a more abstract description (in whatever formalism is being used) will have a larger set of possible world states that satisfy it. When less abstract descriptions are added, the size of this satisfying set diminishes as things in the world are discriminated in increasingly finer detail. In complex worlds, these abstract descriptions can often be idealizations. This means that a plan realizable at an abstract level may not be realizable in a finer grain (i.e., the satisfying set might reduce to the null set). For example, one might ignore friction in an abstraction of the domain, but find that the abstract plan cannot be achieved at the lower abstraction level when the effects of friction are included in the world description.

To see how hierarchical planning can help avoid the combinatorial explosion involved in reasoning about primitive actions, consider planning to build a house. At the highest abstraction level might be such steps as site preparation and foundation laying. The

planner can plan the sequence of these steps without considering the detailed actions of hammering a nail or opening a bag of cement. Each of these steps can be expanded into more detailed actions, the most primitive of which might be nail-driving and wire-cutting. Hierarchical abstraction levels provide the structure necessary for generating complex plans at the primitive level.

2 The Many Guises of Hierarchical Planning

The planning literature has used the term "hierarchical planning" not only to describe levels of abstraction, but also to describe systems containing various hierarchical structures or search spaces, metalevels, and what we will call *planning levels.* Examples of each of these are given below. Planning levels are of particular importance because confusing them with abstraction levels causes a problem in various implementations of hierarchical planning that will be discussed in the remainder of this paper.

Many planners produce hierarchical structures (e.g., subgoal structures) during the planning process or explore hierarchically structured search spaces. Generally having nothing to do with abstraction levels, they occur even in nonhierarchical (by our definition) planners that allow only one level of abstraction. STRIPS [1], while nonhierarchical, could be regarded as producing plans with a hierarchical structure, e.g., its triangle tables. The Hayes-Roths [2] use the term *hierarchical* to refer to a top-down search of the space of possible plans where more abstract plans are at the top of this search space. This involves a hierarchical search space that contains abstraction levels, but the levels in the hierarchy are not defined by these abstraction levels.

Hierarchical planning is also used to refer to metaplanning. Reasoning at a metalevel involves reasoning about the planning process itself. This is an entirely different domain,

not merely an abstraction or idealization of the original domain. Stefik [7] states that ". . .layers of control (termed *planning spaces*) . . . are used to model hierarchical planning in MOLGEN". In this case, the three planning spaces are being used to implement metaplanning and, respectively, represent knowledge about strategy, plans and genetics; the first two are not abstractions of the genetics domain. (MOLGEN does provide for planning at different levels of abstraction through its constraints.)

The above uses of the term "hierarchical planning" describe processes or structures unrelated to the use of abstraction levels. Therefore, any confusion generated is terminological and is not an indication of possible conceptual problems within the planning system itself. Planning levels, on the other hand, have been confused with abstraction levels – which, as we shall see, can lead to problems within the planner, particularly if the planner incorporates the STRIPS assumption [9]. *Planning levels* are artifacts of particular planning systems and may vary considerably from planner to planner. They are not defined by a different level of abstraction in the descriptions being manipulated, but rather by some process in the planning system. Most planning systems have some central iterative loop that performs some computation on the plan during each iteration. This may involve applying schemas, axioms, or operators to each element of the existing plan to produce a more detailed plan. To the extent that such an iteration takes one well-defined plan and produces another well-defined plan, we will call it a planning level. Planning levels may correspond, in some systems, exactly to the hierarchical structures discussed earlier, but this is coincidence. They are defined by the planning process, not data structures, and may or may not correspond to hierarchical data structures within a particular system. In general, the term is admittedly vague, but in many AI planning systems of interest it has a very precise definition.

In particular, all planning systems in the NOAH tradition, including SIPE [10], NOAH

[6], NONLIN [8], and ABSTRIPS [5], have distinct and well-defined planning levels. In these systems, a new planning level is created by expanding each node in the plan with one of the operators that describe actions. In the literature these levels are often referred to as hierarchical, which implicitly associates them with abstraction levels. In fact, they are independent of abstraction level; a new planning level may or may not result in a new abstraction level, depending upon which operators are applied. For example, in the blocks world described by Sacerdoti [6], there is only one abstraction level. CLEARTOP and ON are the only predicates and each new planning level simply further specifies a plan involving these predicates. Thus, all the hierarchical levels in the NOAH blocks world are actually planning levels, that result in adding further detail to the plan at the same abstraction level. Others have described such an omission of detail as an "abstraction", but we specifically require an abstraction level to involve different predicates with different grain-sizes.

Planning systems not in the NOAH tradition also have planning levels. Rosenschein's planning algorithm, using dynamic logic [4], attempts to satisfy a set of planning constraints. Running his "bigression algorithm" on each constraint in the set (which he claims is a straightforward extension of his system) would constitute a planning level. Agenda-based planning systems also have natural planning levels that are defined by the execution of one agenda item. These planning levels would be somewhat different from the others we have discussed, as they might not involve performing some operation on each element of a complete plan. Consequently, they are not likely to be confused with abstraction levels.

3 A Problem Encountered with Current Planners

There is a problem that can arise when planning and abstraction levels are interleaved in a planner making the STRIPS assumption, as they are in the aforementioned systems. (The

STRIPS assumption states that things remain unchanged during an action unless specified otherwise.) This problem exists in many NOAH-tradition planners but has never been documented.

In such a planner, one element of the plan can attain a lower abstraction level than another element at the same planning level, depending upon which operators are applied. Thus, the plan at the current planning level could be **P1;Q1;G**, where **P1** is an action making the predicate P true, **Q1** an action making the more abstract predicate Q true, and **G** a goal that depends upon the truth of P for its achievement. With the STRIPS assumption, the planning system will find that P is true at **G**, since **Q1** does not mention changes involving any less abstract predicates (such as P). In fact, the truth of P may depend upon how **Q1** is expanded to the lower abstraction level, since it may or may not negate the truth of P. Thus, conditions may be evaluated improperly in these planners, resulting in incorrect operator applications.

For these planners to correctly test the truth of a condition at time N in the plan, they must ensure either that all relevant information at the proper abstraction level is available for the actions preceding N, or that subsequent expansions to lower abstraction levels will not change the truth value of the condition. However, many existing planners (e.g., NOAH and SIPE) do not provide this assurance. There are good reasons for this. Various solutions to the above problem are discussed later in this paper, but the most straightforward one is to impose a depth-first, left-to-right planning order that is sensitive to change in abstraction level. This means that, when a condition with predicates of abstraction level M is checked at time N in the plan, all possibly relevant information at abstraction level M or higher will be available for every plan element occurring prior to N.

However, this is not always desirable since many advantages can be gained by planning certain parts of the plan expedientially (or opportunistically) to lower abstraction levels.

For example, in planning a trip from Palo Alto to New York City, it might be best to plan the details of the stay in New York first, as this could determine which airport would be best to fly into, which in turn could determine which Bay Area airport would be the best departure point. Thus, we do not want to restrict ourselves to depth-first, left-to-right planning, which would require choosing the Bay Area airport before the one in New York.

While the foregoing problem will be discussed in this paper in the NOAH-like terminology of "operators" and "goals", it applies equally well to any planner that must coordinate deductions over different abstraction levels. For example, Rosenschein's hierarchical planner based on dynamic logic [4] must address this issue in order to produce correct plans.

The definition of this problem will be made more concrete by looking at it in the context of an indoor robot domain, which will be used for expository purposes throughout the rest of this paper.

3.1 Abstraction Levels in an Indoor Robot Domain

We recently encoded a simple indoor robot world in the SIPE planning system that revealed the problem of coordinating abstraction levels. The robot domain consists of 5 rooms connected by a hallway in the Artificial Intelligence Center at SRI International, the robot itself, and various objects. The rooms were divided into 35 symbolic locations that included multiple paths between locations (which greatly increases the amount of work done by the planner). The initial world was described by 222 predicates, about half of which were deduced from SIPE's deductive operators. The description of possible actions in SIPE included 25 operators describing actions and 25 deductive operators. The operators use four levels of abstraction in the planning process. The planner produces primitive plans that provide actual commands for controlling the robot's motors.

The most abstract level of the planning process reasons about the tasks that can be

performed, such as preparing a report or delivering an object. Our simple domain requires only one level for this, but more complex tasks might require several abstraction levels to describe them. The first level below the task level (referred to as the INROOM level since INROOM is the crucial predicate) is the planning of navigation from room to room. This plans a route that may require many planning levels for all the necessary operators to be applied, but it does not involve any reasoning about particular doors or locations. High-level predicates describing connections indicate that it is reasonable to move from one room to another, but without first considering any details as to how this might be done or whether it might even be possible in the current situation. When such a move is planned to a lower abstraction level, it may fail or many actions may have to be performed to clear a path.

Below the room level is the NEXT-TO level, which plans movements from one important object (that the robot is next to) to another. For example, to copy a paper, the robot will have to get next to the door of the copy center, then pass through the doorway, then get next to the desk of the operator, etc. This abstraction level plans high-level movements within a room but is still not concerned with actual locations. NEXTTO is the crucial predicate at this abstraction level.

The lowest level is the location level, where SIPE plans movements down to the level of the actual locational grid it has been given. This may involve planning to move obstacles so as to clear particular paths. AT is the crucial predicate at this abstraction level.

3.2 Coordinating Abstraction Levels

The problem of coordinating abstraction and planning levels arose during planning in the robot domain in the plan depicted in Figure 1. The initial goal produced three INROOM goals at the first planning level. Of these three subtrees, the first and last were transformed

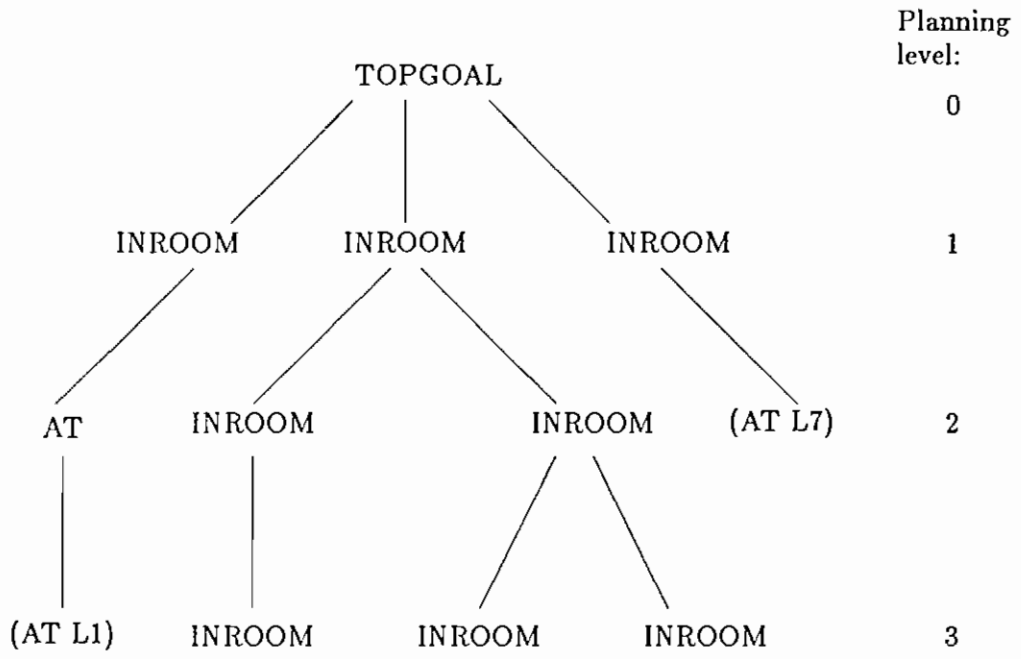

Figure 1: Hierarchical Plan in Robot Domain

into the lowest abstraction level at planning level 2 (intervening NEXTTO goals have been disregarded for the sake of simplicity). The middle subtree, however, is still at the room abstraction level on planning level 3, as it required several operator applications to find a path through the rooms to accomplish its goal.

Now, when the planner applies an operator to the (AT L7) goal at planning level 2, any precondition that queries the AT predicate will find (AT L1) to be valid, since that is the last place at which AT is affected and the STRIPS assumption assumes all actions leave predicates unchanged unless they explicitly specify otherwise. But this is not correct, as AT may be affected after the middle subtree is expanded to the lowest abstraction level. If the operator depends critically on the value of the AT predicate, its application to (AT L7) may be incorrect. The resulting plan will have commands that move the robot from location L1 to location L7, whereas the robot is not likely to be at L1 when this part

of the plan is executed (because of movements made in the middle subtree). Whether such an operator application will prevent the correct plan from being found depends on how a particular planner detects invalid plans and how the search space is organized.

4 Solutions

There are many alternative solutions to the problem discussed above. They range from calculating all possibly relevant information before it may be needed to ignoring the problem altogether and simply letting the user beware of the consequences. The former, and most straightforward, is to force the planner to plan in temporal order and to provide for calculation of all the predicates at one abstraction level that may be needed later. Many planners do this, though it is usually not made clear that they depend on several assumptions to avoid subtle problems such as this one. ABSTRIPS [5] is an example of planners that use this approach. It assigns abstraction-level numbers to the predicates and plans a lower level only when all necessary computations have been made. This could easily be implemented in systems like NOAH and SIPE.

The problem with this approach is that the planner is limited in the order in which it can process goals. Quite often the order imposed will not be optimal (as in the Palo Alto-to-New York example). The flexibility to plan certain parts of the plan expedientially to lower abstraction levels is lost. Constraints generated during such lower-level planning can narrow down the search, thus resulting in potentially large gains in efficiency. For these reasons, planners like SIPE, NOAH, and NONLIN allow the mixing of planning and abstraction levels.

The approach used in these latter planners is susceptible to the level coordination problem and therefore requires the user to be alert. Incorrect checking of conditions similar

to the one described can occur in these systems, depending on the task and the encoding of the operators. This can result in incorrect operator applications. The planning system may have mechanisms that later detect the plan is incorrect, as SIPE and perhaps NONLIN do. However, the proper solution may not be found if the operator applied incorrectly is the one actually needed for the correct solution, since it may not be retried. Its premature application should have been delayed one or more planning levels so that other parts of the plan could be planned to a lower abstraction level. The user is responsible for writing operators that will accomplish this delay. This process is facilitated by certain features of SIPE, as we shall see.

It is appealing to look for a technique between the inefficiency of computing everything in a certain order, and the expediential behavior of systems that perhaps miss valid solutions. This would involve reasoning about what properties will remain invariant during further planning of certain goals. For example, in Figure 1 the planner might calculate whether the possible expansions of the middle subtree will affect the value of the AT predicate. If not, planning can proceed expedientially. This is all the more appealing because reasoning about concurrency depends on determining invariant properties of a sequence of actions.

Despite the attraction of this approach, there are severe difficulties entailed in computing these invariances. It would be best if they could be computed automatically from the operators without requiring the user to supply additional knowledge. (The STRIPS assumption effectively makes this computation for predicates at one abstraction level, but the computation must be done for predicates at other abstraction levels.) In general, this is not computationally tractable. It is similar to the problem of regressing conditions through actions ([4],[9]), except that the regression must be done through every possible expansion of the actions for an indeterminate number of planning levels. Furthermore,

simple schemes that simply check for predicate names that have possibly been changed will probably find very few invariances. For example, in the robot domain, every high-level goal will alter the values of AT predicates at the lowest level. More sophisticated schemes that check possible values for the arguments of the predicates, perhaps determining ranges of values they might acquire in all possible expansions, would themselves require solving a large search problem.

An alternative is to have the user provide information about what remains invariant over actions. While this may be useful for some domains, in general the same criticisms made above apply to this case. There will in general not be many things (at lower abstraction levels) that are invariant; moreover, it may even be difficult to indicate explicitly what they are, as the invariance may involve complex constraints on the allowable arguments to predicates. In addition, one of the chief advantages of abstraction levels is that specifying details is unnecessary at higher levels. Computing invariants would require information as to which lower levels are affected in what way by each higher-level goal, thus removing some of the advantage gained by not planning at the lowest level from the very beginning. The computational costs of this can quickly become overwhelming as we shall see in the next section.

4.1 Delaying Operator Applications in SIPE

For reasons given above, SIPE places the burden on the user to encode the domain in such a way that incorrect evaluation of conditions will not produce applications of operators that prevent solutions from being found. We have solved this problem within SIPE for the indoor robot domain in two different ways. One solution involves delaying the application of certain operators and the other solution involves the introduction of certain less abstract predicates at earlier planning levels.

```
OPERATOR: not-yet
ARGUMENTS: robot1,location2,area1,location1;
PURPOSE: (at robot1 location2);
PRECONDITION: (at robot1 location1),
     (inroom robot1 area1),
     (not (contains location1 area1));
PLOT: COPY
END PLOT END OPERATOR
```

Figure 2: Operator for Delaying Operator Application

The first solution involves a novel use of operators, developed during implementation of the robot domain, that effectively delays the achievement of certain goals until the appropriate juncture, as long as the latter can be ascertained by conditions that are expressible as preconditions of a SIPE operator. This is best shown by returning to the robot domain example.

In the robot domain, only the planning of AT goals is affected when abstraction levels varied in the plan. Figure 1 depicts the type of situation in which the accomplishment of an AT goal must be delayed. This is done in SIPE by using the operator shown in Figure 2. It delays the solving of AT goals until the part of the plan preceding them has been brought to the same level of abstraction. This is done by checking whether the AT location of the robot is in the same room as its INROOM location. If the precondition of this operator matches, it means that the last AT predicate specified as an effect of an action came before the last INROOM predicate specified as an effect. Consequently, the latter action must still be planned to the lower level of abstraction. [1]

[1] Of course, this operator could still be fooled if you planned a circular route that ended in an INROOM goal for the same room that contained the last preceding AT location. However, this would cause a problem only if the eventual location reached in the expansion of the INROOM goal were different from the one in the earlier AT goal. This situation never arises in our domain.

This operator is applied before any other to an AT goal. The plot of not-yet is simply the token *COPY* that copies the goal from the preceding planning level. It is necessary to use a special token rather than specify the AT goal in normal syntax. Normally SIPE inserts the precondition of an operator into the plan and maintains its truth. In this case the precondition will not be true in the final plan, so the COPY option inserts the appropriate goal without first inserting the precondition. With this feature and the above operator, SIPE can mix abstraction and planning levels freely in the robot domain without missing a solution on our test problems. However, there are problems in the domain that cannot be solved without creating additional delaying operators.

4.2 Introducing Low-Level Predicates in SIPE

Instead of using the not-yet operator, the second solution involves introducing lower-level predicates at higher abstraction levels in order to prevent the STRIPS assumption from causing a problem. In this case, we add a lower-level AT predicate (which includes an uninstantiated locational variable) to every higher-level NEXT-TO goal as a placeholder for the predicate that would be produced during some future expansion. When the NEXT-TO goal is expanded to the lower level, the location actually reached will eventually become the instantiation of the locational variable introduced. At the planning level of the NEXT-TO goal, any AT predicate in a condition being tested will match with the newly inserted AT predicate, preventing the planner's incorrect assumption that the NEXT-TO goal does not affect the truth-value of the condition.

This solution takes advantage of SIPE's ability to post constraints on variables. The newly introduced AT predicates effectively document the fact that the AT location may eventually change during any expansion of the NEXT-TO goal (even though the location is not yet known). Before and during such an expansion, the location variables can accu-

mulate constraints on their possible values, so the planning process will not be hindered. The matching of conditions will always be correct because these AT predicates are present everywhere the AT location might change.

Incorporating this change into the SIPE operators written for the first solution was easy. Only three operators of the 25 posted NEXT-TO goals, so only those three had to be changed. The process of converting these three operators is illustrated by contrasting the original FETCH operator shown in Figure 3 with the FETCH operator including AT predicates shown in Figure 4. In the plot, each goal and process with a NEXT-TO predicate in its effects is given an additional effect that is an AT predicate involving a new locational variable. The latter is included in the arguments of the goal or process so as to permit appropriate matching with the variables in the operators that solve NEXT-TO goals. The locational variable is added to the arguments of the operator, whose precondition can also specify any predicate that constrains the variable. At a minimum, the variable can be constrained by its containing room; stronger constraints can sometimes be specified.

Once the three operators were converted in this manner, SIPE was able to solve all the problems in our test domain. This solution appears robust and should not prevent the planner from successfully dealing with any problems it might solve by means of an ABSTRIPS-like approach. Characteristics of the domain are again exploited in this solution. By having to plan about less abstract entities at a more abstract level, we are giving up some of the advantage gained by planning hierarchically. However, it is reasonable in this case because we need introduce only one lower level predicate early (albeit the most important one), and we need introduce it only a single abstraction level early. There is no difficulty in coordinating any other pair of abstraction levels in this domain. However, the introduction of more variables and constraints significantly increases the effort required. Using this technique to solve a problem that entails seven planning levels for producing

```
OPERATOR: fetch
ARGUMENTS: robot1,object1,area1;
PURPOSE: (holding robot1 object1);
PRECONDITION: (inroom object1 area1);
PLOT:
    GOAL: (inroom robot1 area1);
        MAINSTEP: (holding robot1 object1);
    GOAL: (nextto robot1 object1);
    PROCESS
        ACTION: pickup;
        ARGUMENTS: robot1, object1;
        EFFECTS: (holding robot1 object1);
END PLOT END OPERATOR
```

Figure 3: Original FETCH Operator

```
OPERATOR: fetch
ARGUMENTS: robot1,object1,area1,location1;
PURPOSE: (holding robot1 object1);
PRECONDITION: (inroom object1 area1),
    (contains location1 area1);
PLOT:
    GOAL: (inroom robot1 area1);
        MAINSTEP: (holding robot1 object1);
    GOAL: (nextto robot1 object1);
        ARGUMENTS: robot1, object1, area1, location1;
        EFFECTS: (at robot1 location1);
    PROCESS
        ACTION: pickup;
        ARGUMENTS: robot1, object1;
        EFFECTS: (holding robot1 object1);
END PLOT END OPERATOR
```

Figure 4: FETCH Operator with AT Predicate

a primitive plan with 58 process/phantom nodes takes roughly twice as long as using the not-yet operator (60 to 80 seconds of CPU time on a Symbolics 3600, rather than just 35).

4.3 Comparison of Solutions

The first solution accomplishes the delayed application of operators when necessary, but permits expediential planning in other cases. This retains flexibility while remaining efficient by not regressing conditions through possible expansions of actions. As no lower-level predicates are introduced early, full advantage is taken of hierarchical planning. The disadvantages of this approach are that the user (though relieved of the necessity of specifying invariance properties for higher-level goals) must write appropriate delaying operators and, furthermore, must have anticipated all possible situations in which operators would need to be delayed. In complex worlds this means that novel problems might not be solvable. In addition, it may not always be possible to express the appropriate delaying conditions as a SIPE precondition. In an application in which efficiency is of paramount importance and failure to solve a particular problem can be tolerated, this may be a desirable approach.

The second solution is more robust and less likely to fail on novel problems. It was surprisingly easy to implement in SIPE. However, when low-level predicates are introduced at a higher abstraction level, it is significantly less efficient. The advantages of hierarchical planning can be readily seen, as the introduction of only one predicate (albeit the crucial one) at the next higher abstraction level approximately doubles the cost of computation.

5 Summary

Ambiguities in the planning literature involving hierarchical levels are explicated. The interleaving of planning levels and abstraction levels prevents some current planners from

finding correct solutions. Various techniques designed to prevent this have certain drawbacks; two methods implemented in SIPE involve different tradeoffs between flexibility, efficiency, and robustness.

References

[1] Fikes, R. E. and Nilsson, N. J., "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving", *Artificial Intelligence 2*, 1971, pp. 189-208.

[2] Hayes-Roth, B. and Hayes-Roth, F., "A Cognitive Model of Planning", *Cognitive Science 3*, 1979, pp. 275–310.

[3] Hobbs, J.. "Granularity", *Proceedings IJCAI-85*, Los Angeles, California, 1985, pp. 432-435.

[4] Rosenschein, S., "Plan Synthesis: A Logical Perspective", *Proceedings IJCAI-81*, Vancouver, British Columbia, 1981, pp. 331-337.

[5] Sacerdoti, E., "Planning in a Hierarchy of Abstraction Spaces", *Artificial Intelligence 5 (2)*, 1974, pp. 115-135.

[6] Sacerdoti, E., *A Structure for Plans and Behavior*, Elsevier, North-Holland, New York, 1977.

[7] Stefik, M., "Planning and Metaplanning", in *Readings in Artificial Intelligence*, Nilsson and Webber, eds., Tioga Publishing, Palo Alto, California, 1981, pp. 272–286.

[8] Tate, A., "Generating Project Networks", *Proceedings IJCAI-77*, Cambridge, Massachusetts, 1977, pp. 888-893.

[9] Waldinger, R., "Achieving Several Goals Simultaneously", in *Readings in Artificial Intelligence*, Nilsson and Webber, eds., Tioga Publishing, Palo Alto, California, 1981, pp. 250–271.

[10] Wilkins, D., "Domain-independent Planning: Representation and Plan Generation", *Artificial Intelligence 22*, April 1984, pp. 269-301.