Report Number: WUCS-83-3

1983-05-01

Fuzzy Algorithms, Planning and Problem Solving

Stuart Goldkind

For the last decade the STRIPS paradigm has had a dominant influence on research in planning systems. More recently researchers have realized that a departure from this framework is necessary if planning systems are to be able to deal with complex, real-world environments in an intelligent, flexible, and efficient manner. The present report surveys some of the problems encountered by workers in this area, and indicates what the author believes are some of the underlying causes of the difficulties. The author then proposes the notion of a "fuzzy algorithm" as a means for encorporating strategy and flexibility into the... **Read complete abstract on page 2.**

Fuzzy Algorithms, Planning and Problem Solving

Stuart Goldkind

Complete Abstract:

For the last decade the STRIPS paradigm has had a dominant influence on research in planning systems. More recently researchers have realized that a departure from this framework is necessary if planning systems are to be able to deal with complex, real-world environments in an intelligent, flexible, and efficient manner. The present report surveys some of the problems encountered by workers in this area, and indicates what the author believes are some of the underlying causes of the difficulties. The author then proposes the notion of a "fuzzy algorithm" as a means for encorporating strategy and flexibility into the planning process. It is suggested that that the further development of the idea of a "fuzzy algorithm" is a fruitful direction for research which can provide a unifying framework for various other proposals encountered in the literature.

FUZZY ALGORITHMS, PLANNING AND PROBLEM SOLVING

Stuart Goldkind

WUCS-83-3

Department of Computer Science

Washington University

St. Louis, Missouri 63130

May, 1983

ABSTRACT

For the last decade the STRIPS paradigm has had a dominant influence on research in planning systems. More recently researchers have realized that a departure from this framework is necessary if planning systems are to be able to deal with complex, real-world environments in an intelligent, flexible, and efficient manner. The present report surveys some of the problems encountered by workers in this area, and indicates what the author believes are some of the underlying causes of the difficulties. The author then proposes the notion of a "fuzzy algorithm" as a means for encorporating strategy and flexibility into the planning process. It is suggested that the further developement of the idea of a "fuzzy algorithm" is a fruitful direction for research which can provide a unifying framework for various other proposals encountered in the literature.

CONTENTS

FUZZY ALGORITHMS, PLANNING, AND PROBLEM SOLVING

I. Introduction.

While there has been no general agreement on exact definitions of basic terms (e.g. exactly what constitutes a "plan," "problem," "solution," etc.) [McDermott 1978] one general framework has had a dominant influence on research in planning systems over the last decade. This is the paradigm based on the STRIPS system (Stanford Research Institute Problem Solver) [Fikes, Nilsson 1971; Fikes 1971; Fikes, Hart, Nilsson 1972]. In the present report I will refer to systems based on this paradigm as SL systems (SL = STRIPS-Like). The only claim being made in referring to a system as SL is that some significant subset of the assumptions embodied in the STRIPS paradigm are shared by the system in question. An SL system may (and usually does) differ conspicuously from the original STRIPS system. The STRIPS paradigm shares with any SL system the following general features:

(1) A mechanism for world modeling
(2) A set of operators on world models
(3) Mechanisms for producing and selecting paths in a search space generated by operator applications on world models.

In STRIPS the world modeling is done with FOL (first order logic), a world model being a state description. Each operator consists of a set of preconditions, an add list, and a delete list. These "operators" were introduced in order to avoid difficulties with the "frame problem" (see below under the heading "Epistemological Non-determinism"). The preconditions are a set of FOL wffs (well formed formulas) giving the conditions under which the operator is applicable, while the add and delete lists specify the effects of

the operator. The add list gives those wffs which must be added to a new state description that results from the application of the operator to a given state description; the delete list indicates wffs which will no longer hold in the new state description and therefore must be deleted. Strictly speaking, of course, the operators are general schema whose parameters must be properly instantiated to fit a given state description. For example:

State description: CLEAR(B) CLEAR(C)
ON(C,A) HANDEMPTY
ONTABLE(A) ONTABLE(B)

corresponds to the picture:

FIGURE 1.

Operators:

(1) PICKUP(X)

Preconditions: ONTABLE(X), CLEAR(X), HANDEMPTY

Add: HOLDING(X)

Delete: ONTABLE(X), CLEAR(X), HANDEMPTY

(2) PUTDOWN(X)

Preconditions: HOLDING(X)

Add: ONTABLE(X), CLEAR(X), HANDEMPTY

Delete: HOLDING(X)

(3) STACK(X,Y)

Preconditions: HOLDING(X), CLEAR(Y)

Add: HANDEMPTY, ON(X,Y), CLEAR(X)

Delete: HOLDING(X), CLEAR(Y)

(4) UNSTACK(X,Y)

Preconditions: HANDEMPTY, CLEAR(X), ON(X,Y)

Add: HOLDING(X), CLEAR(Y)

Delete: HANDEMPTY, CLEAR(X), ON(X,Y)

In the world given by the state description and depicted in Figure 1, PICKUP(B) is applicable, as is UNSTACK(C). PICKUP(A), on the other hand, is inapplicable, since when we instantiate X to A we find one of the preconditions of PICKUP(A) is not satisfied; viz. it is not the case that CLEAR(A) (since C is on A).

STRIPS' method of finding a solution path through the search space is derived from that of GPS [Ernst, G. and Newell, A 1969; Fikes and Nilsson 1971]. First a set of "differences" between the present world model and the goal is extracted; then a set of operators "relevant" to reducing those differences are identified. Once a relevant operator is selected, the original goal is replaced by the new (sub)goal of reaching a state where that operator is applicable. When such a state is found, then the relevant operator is applied and the original goal is reconsidered in the resulting state description.

There are, of course, many different possible strategies for finding a solution path and many elaborations on those strategies (forward and backward chaining combined with heuristics, goal regression, parallel planning, hierarchical planning, etc.). However, the basic ideas of what constitutes a plan and how a plan is arrived at remain the same for SL systems: The set of operators, taken together with the initial state, give us a search space; and the description of the goal state determines a set of solution paths, one of which must be discovered by the planner. The resulting notion of a plan is that of a sequence of operators (actions) which leads from the initial state to the goal state.

That there are serious problems with using this sort of model of planning is not a surprise at this time. See notably: [McDermott 1978; Sacerdoti 1979; Wilensky 1981a]. The STRIPS paradigm as outlined above is not the way for a robot (or a human) to deal with complex, nonstatic, real world environments. There is little agreement, however, as to the causes of the inadequacies of the SL approach in this regard, nor is there any clear remedy. Everyone would like robots to be able to deal with complex, changing, real environments [Sacerdoti 1979], but we are only beginning to understand the problems involved. In the rest of this report I try to shed light on some of the underlying causes of these difficulties and indicate fruitful directions for further research.

II. Post-Strips Problems.

Much of the research after STRIPS has been devoted to finding solutions for problems which arise in SL systems. The result has been a veritable garden of interesting variations on the basic SL strain. Many of the problems investigated are not specific to SL systems, and must be solved by any system that uses plans formulated on the basis of beliefs about future states of the world. As we shall see, however, some of these problems are not best addressed within the framework of SL systems. In this section we delineate two of the most widely recognized problems in planning research, "subgoal interaction" and "combinatorial complexity"; in the sections following this one we explore some of the reasons for supposing that a new approach is likely to yield useful results with respect to these and other problems.

(A) Subgoal Interaction.

> One of the subtler issues of automatic problem solving and program synthesis is subgoal annihilation or infringement. Subgoal annihilation occurs when during the course of solving one aspect of a larger problem, some other aspect of the problem whose solution has already been obtained is interfered with, or undone completely. Subgoal infringement is a milder form of subgoal annihilation in which the prior solution of one subgoal causes the solution of another later subgoal to become more difficult or tedious. [Rieger and London 1977]

One of the most famous proposals for a solution to the problem of subgaol annihilation is one by Earl Sacerdoti [Sacerdoti 1973, 1975, 1977]. He calls attention to a problem presented by [Sussman 1975] . We will refer to this

problem as "Example 1".

The problem can be illustrated quite simply if we take as given the initial position of Figure 1 above and start with the goal ON(A,B) & ON(B,C). To simplify we assume that the only operator is:

```
PUTON(X,Y)
 Preconditions:  CLEAR(X) CLEAR(Y) ON(X,Z)
           Add:  ON(X,Y)
        Delete:  CLEAR(Y) ON(X,Z)
```

The first step for any SL system is to decompose the original conjunctive goal into two subgoals, one for each conjunct. The system will then try to achieve each of the conjuncts in turn. Suppose it tries to achieve ON(A,B) first. After clearing A by applying the operator (performing the action) PUTON(C,TABLE), it can achieve ON(A,B) by doing PUTON(A,B). The result is then:

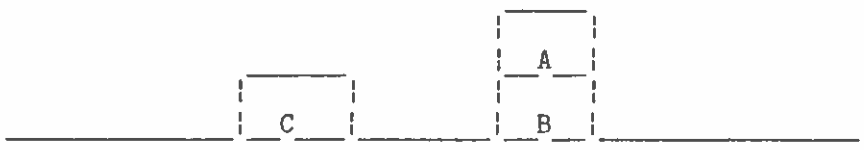

FIGURE 2

But now, in order to achieve ON(B,C), it will have to first clear B by PUTON(A,TABLE). This, however, undoes the subgoal which it had already achieved. On the other hand, if the system tries to achieve ON(B,C) first, it can immediately do PUTON(B,C), resulting in:

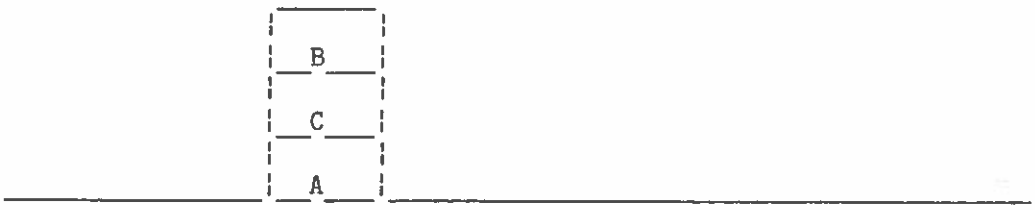

FIGURE 3

But now the system is even farther from its goal than it was in the initial state, and to make any progress it would again have to undo a subgoal that it had previously achieved.

In this example undoing the subgoals might seem relatively harmless. After all, in Figure 2, if the system does PUTON(A,TABLE) (and assuming that it avoids repeating states) it could then do PUTON(B,C) followed by PUTON(A,B), thus achieving the goal ON(A,B) & ON(B,C). The difficulty is that all problems are not this easy, and in a more complex domain the relatively unintelligent process of annihilating and re-achieving arbitrary subgoals is as undesirable as it is inefficient. Early researchers had this in mind when they introduced the relatively simple idea of subgoal protection [Waldinger 1975; London 1978b]. Subgoal protection simply means, as the name implies, that once a subgoal has been achieved it must be protected from annihilation by subsequent actions (operator applications). Whatever device is used to accomplish subgoal protection, however, there always arise cases of conflict between the principle of subgoal protection and the need to explore different solution paths. Thus if subgoal protection is applied without restriction, the system cannot achieve the goal of Example 1 at all (since once one of the subgoals is reached, the system is effectively stuck).

Sacerdoti's proposed solution is for plans to be represented as partial orderings during the initial stages of their construction. The system can begin with an over-simplified plan that considers the subgoals of achieving ON(A,B) and ON(B,C) as parallel and independent operations. Once the seperate parallel subplans for these subgoals are complete a set of "critics" can analyse the partially ordered plan for possible conflicts.

Sacerdoti illustrates his system with a plan involving painting. The system starts with the goal "paint the ceiling and paint the ladder". This goal is decomposed into the two subgoals "paint the ceiling" and "paint the ladder," and then two parallel plans are developed to achieve each subgoal independently. The two (parallel) plans are:

(get paint, get ladder, apply paint to ceiling) and
(get paint, apply paint to ladder).

A potential conflict is then observed between "apply paint to ladder" and "apply paint to ceiling". This is because "apply paint to ceiling" presupposes "has ladder" (which would be achieved by "get ladder"), but "apply paint to ladder" deletes "has ladder". The system avoids the possible conflict by imposing the constraint that "apply paint to ceiling" must preceed "apply paint to ladder" [Sacerdoti 1975, 1977].

While Sacerdoti's work on parallel planning is generally recognized as first rate, and is cited by Nils Nilsson as being a "milestone in automatic planning" (Nilsson in foreword to [Sacerdoti 1977]) there remain difficult issues to resolve. How, for example does the system "know" that "apply paint

to ladder" deletes "has ladder"? According to Sacerdoti's description, "has ladder" is on the delete list of "apply paint to ladder," but this is ad hoc, at best. It is simply not true that "has ladder" should be deleted by "apply paint to ladder"; the system (or whoever painted the ladder) still has a ladder, viz. he (or it) now has a painted ladder. And, in fact he could still use the ladder to paint the ceiling if painting the ceiling were more important than ruining the paint job on the ladder or if he didn't care much about the quality of the paint job on the ladder (not to mention the possibility of waiting until the ladder was dry; or a case where someone is stuck on the roof).

The point is not, of course, that the delete list of one particular operator is wrong (nor that using the add-list-delete-list form of operator is limited, although we will see that this is true in the subsequent discussion). There is a fairly deep problem here that is still under investigation [London 1978b; Stefik 1981a; Wilensky 1979,1980,1981; McDermott 1978]. It is simply not the case that all (sub)goal conflicts can be directly deduced from the add and delete lists of the primative operators. Rieger and London give the following example which shows that subgoal conflicts are not even always due to ordering problems:

> ..if our AI researcher wishes to insert a stake in his garden for the purpose of nailing a cookie tin to its top as a bird feeder, selecting and applying the "pound it in with a sledgehammer" strategy might severely splinter the top of the stake, making it impossible or difficult to nail anything to the stake. [Rieger and London 1977]

So one problem is determining the relative importance of side effects.

Another interesting problem is how to organise the system's knowledge about side effects. That is, if the knowledge that painting the ladder often makes it undesirable to use the ladder (along with all the other bits of possibly but not necessarily relevant information about ladders) is not included in the delete list of the operator "apply paint to ladder" (or if the operator itself were the less ad hoc "apply paint"), where is this information to be stored, and can it even be accomodated by an SL system? Some researchers see a departure from the SL framework in the direction of meta-planning as a possible solution [Wilensky 1981a; McDermott 1978].

Many other ways of dealing with subgoal interaction have been proposed. An idea of Waldinger's has the added attractiveness of having direct applications in automatic programming. The basic mechanism used by Waldinger is called "goal regression," and is a form of plan (or program) modification [Waldinger 1975]. The basic strategy for achieving a conjunctive goal P & Q is to first form a plan which achieves P, and then modify this plan so that it also achieves Q. The reason Waldinger's particular technique is called "regression" is that the goal is "passed back" over stages of the plan, starting from the final state and progressing (regressing) towards the initial state. In general , let P be a predicate and F an operator (or in Waldinger's original case, F is a program instruction). Suppose P is true, then, if we perform the action corresponding to F, there is no guarantee that P will remain true. However, it is always possible (in principle) to find a predicate P' which, if true before F is performed, will guarantee the truth of P after F is executed [Manna and Waldinger 1974; Waldinger 1975]. For example, suppose P is CLEAR(C) and F is PUTON(A,B), then P' is CLEAR(C) v ON(A,C). Or, (Waldinger's programming example) let F be (X := t) where X is a

variable and t is an expression, and let P be any restriction on the variables of the program, written: P(X). Then P' is P(t) or P(X){[X/t]} (i.e. the result of replacing X by t in P(X)). For example, F is (X := U * V) and P(X) is (X = A * B). then P' is (U * V = A * B). Thus, in order to achieve P & Q, we first construct a plan to achieve P, and then "pass back" or "regress" the goal Q over the plan for achieving P. (See [Waldinger 1975] for a more complete account).

Goal regression is quite effective for limited, deterministic domains with complete information. This is also true of a host of other methods for dealing with subgoal interaction. (Subgoal interaction remains a subject of research even as recently as [Steffik 1981]). Everyone seems to have their own favorite methods for dealing with this problem (see [London 1978b; Rieger and London 1977]), but almost without exception the "solutions" rely on the basic SL assumptions as being sound. For example, Sacerdoti's solution relies on the ability of the system to construct ordinary SL plans in parallel, and Waldinger's goal regression really is equivalent to a sophisticated combination of forward and backward chaining. Thus these and other SL systems still adhere to the notion of planning as search and of a plan as a predetermined sequence of primative operators. In subsequent sections I will argue that these assumptions effectively nullify supposed "solutions" to the problem of subgoal interaction when the system must deal with realistic, complex, changing environments.

(B) Combinatorial Complexity.

> General purpose problem solvers, such as STRIPS or GPS, must do their work using general purpose search heuristics. Unfortunately, by using such heuristics, it is not possible to solve any reasonably

> complex set of problems in a reasonably complex domain. Regardless of how good such heuristics are at directing search, attempts to traverse a complex problem space can be caught in a combinatorial quagmire. [Sacerdoti 1973]

This is clearest, perhaps, in the domain of chess. If the average game has 45 moves, this is 90 half moves (or 90 ply in the search tree); and if the branching factor is 38 (i.e. the average number of legal moves in any position) then the size of the bottom tier of the search tree is 38 to the 90th power (close to 10 to the 142nd power). The age of the Earth (estimated) is 4.6 billion years or less than 10 to the 18th seconds.

Every planning system has to have some way of getting around this sort of problem. Sadly, most systems seem to have been designed to literally avoid the problem by considering only extremely simple domains, like the blocks world, or a group of three or four rooms with five or six objects to manipulate.

We will be looking, in this section, at another idea of Sacerdoti's [Sacerdoti 1973] which tries to actually deal with the problem instead of ignoring it. We will find that Sacerdoti's method cannot really come to grips with the problem either, imbedded as it is in the STRIPS paradigm. But the basic idea (which actually goes back to [Polya 1945] and was also used to some extent by GPS) may provide the germs of a method which can provide a solution in another framework. This idea (now known under the heading, "hierarchical planning") is found in one form or another throughout the literature [Wilensky 1981a; Steffik 1981a] and is effective in limited domains.

The basic idea is to take a complex statement of a problem and translate it into a simplified description in a representation which abstracts from details and preserves only crucial factors. Given the description in the simpler more abstract problem space, a solution is found, and then mapped back into the more complex space. If the simpler space indeed preserves the crucial features of the original problem space, then the solution which is mapped back should solve the problem in the original space. This idea is easily extended to include not just one abstraction space, but a hierarchy of such spaces, each level having fewer "details" than the one below it. The process is then one of stepwise refinement mapping down one level at a time from some "higher" solution space to some "lower" one.

What Sacerdoti does is to come up with a way of incorporating this idea into an existing SL system viz. STRIPS; the resultant system he calls ABSTRIPS (AB for ABstraction space). The method of incorporation is to form an abstraction space (actually a hierarchy of such spaces) obtained from the ground space of STRIPS operator applications by altering the level of detail in the preconditions of the operators. The precondition wffs in an abstraction space have fewer literals than precondition wffs for the same operators in the ground space. The literals that are left out are thought of as being mere details with respect to the abstraction space. That is, in the abstraction space it is assumed that a simple plan can be found to achieve these literals once a plan has been found which incorporates the more "critical" literals which are included in the abstraction space. To help ABSTRIPS distinguish between levels of detail, each literal within the preconditions of each operator is assigned a number representing its "criticality". ABSTRIPS procedes by a process of sucessive refinements, first

forming a plan at the highest level of criticality (where it needs to consider very few preconditions), then introducing the preconditions from the next level below and incorporating them into the more "abstract" plan.

For example, suppose a plan required that the operator TURNONLAMP(L) be used. And suppose the criticality levels assigned to each literal in the preconditions were as indicated by the number given in braces:

{4} TYPE(L,lamp)
{3} INROOM(L,roomx)
{3} INROOM(robot,roomx)
{2} PLUGGEDIN(L)
{1} NEXTO(robot,L).

At the top level of abstraction, the only precondition for the operator TURNONLAMP that ABSTRIPS would consider would be TYPE(L,lamp). That is, as long as there was a lamp somewhere in the domain, the planner would assume that it could turn it on. At the next level of refinement, the planner would consider the next two preconditions: INROOM(L,roomx) and INROOM(robot,roomx) i.e. it would now have to achieve, as a subgoal, that the lamp and the robot were in the same room, before TURNONLAMP is applicable. The process continues until reaching level 1 [Sacerdoti 1973].

This is an extremely powerful idea, as far as it goes (we will not discuss here the interesting problem of exactly how the criticality rankings are determined) but it does not go far enough. It is true that some amount of abstraction is being done (enough for tremendous savings in efficiency) but, not nearly what is necessary for realistic domains.

The problem, as I see it, is that the "abstraction" process has still left us at the level of the primitive operators of the domain. This is fine for limited domains, where all goals are artificially constrained to be logical combinations of literals found in add and delete lists of the primitive operators of the domain, but as we shall see, there are realistic goals which do not always lend themselves to this sort of characterization.

III. Fuzzy Goals

Sacerdoti's hierarchical planner, ABSTRIPS, is limited in its ability to abstract from the ground space in two crucial ways: (1) the highest abstraction space that it can plan in is still composed of the basic primitive operators of the domain; (2) the goals it considers are constrained to be logical combinations (usually very simple ones --- Sacerdoti's system could not handle disjunctive subgoals like "to paint the ceiling, get paint and either a ladder or a table" [Sacerdoti 1975]) of literals found in the add and delete lists of the domain operators. There are, however, concepts and goals which are not easily expressed as logical combinations of literals from add and delete lists of domain operators.

For example, consider a subgoal often used in chess when opponents have castled on opposite sides of the board (we will refer to this goal as "PAWNSTORM"). The basic idea is to initiate an attack against the enemy king's castled position by advancing one's pawns in the columns near the edge of the board. The pawns serve initially to chase away enemy pieces posted in front of the enemy king (often gaining important tempi in the process), and ultimately to remove the defending pawns from in front of the enemy king (by sacrifice or trade). The removal of the enemy pawns, of course, serves to remove protection from the enemy king and expose him to attack by pieces, which during the pawn storm must be appropriately posted. In addition, the removal of one's own pawns results in greater freedom for one's pieces. The removal of the king rook pawn, for example, will often provide direct access along the opened rook file for one's queen and rooks. During all of this one must do such things as:

(1) spot any target pieces and pawns
(2) note retreat squares for pieces in (1)
(3) calculate the effect of pawns in (1) advancing
(4) do null move analysis (or possibly tree search) to determine the most effective targets and best order of moves
(5) check for enemy interference
(6) monitor any enemy progress (or attacks) on other parts of the board
(7) monitor the success of the plan (the possibility of changing to another plan, the possibility of altering the present plan, etc...

A goal like PAWNSTORM is not easily expressed in terms of logical combinations of literals from add and delete lists of domain operators. What is to be achieved in accomplishing this goal is not any one particular state (or even any one particular sort of state) but, rather, a complicated process which is performed over a large number of states. [McDermott 1978] notes that there are many goals which do not lend themselves to characterization as states of affairs to be brought about. He lists, among others:

"Wait here for five minutes."

"Promise me you will pay me five dollars."

"Avoid firing the budget director while retaining credibility."

"Keep track of the number of sheep on this birch bark."

"Design a circuit which converts a square wave into a sine wave."

I suspect that a stronger claim may be true: it may be that there are goals or concepts which cannot be expressed as logical combinations of the appropriate literals from the domain operators. However, since that would be an extremely difficult claim to prove, in this discussion we will be concerned with the weaker result: viz. that there are concepts and goals which are not easily so expressed. We will refer to such goals and concepts as being "fuzzy" with respect to the domain. (The use of the term "fuzzy" here has no

direct connection to the notion of fuzzy sets or to fuzzy logic, although there may well be strong relationships between those notions and the present ones). The fuzzy concepts, I will claim, serve a useful purpose in human planning, and need somehow to be incorporated into planning systems which deal with realistic domains.

There may be different types of fuzzy concepts; I have not undertaken any classification. One characteristic which many examples seem to share is that they involve the notion of an activity or a process which occurs over an extended period of time. Thus these concepts cut across state descriptions, and do not lend themselves to expression in terms of the truth of predicates in a single state description.

Consider, for example, a more familiar activity than PAWNSTORM; for instance, the problem of planning a picnic. What has to be planned for (as with PAWNSTORM and others activities like it) is not something that happens at just one point in time or in any one particular state description. It is a sustained activity, a sequence of events, which will most likely involve at least a few hours, and perhaps a whole day. This is one aspect of the fuzziness, which is not readily captured by an SL system. Another area of fuzziness for concepts like this one is that it is clear that there is no one rigid set of specifications for what constitutes a picnic. We have already mentioned that the amount of time can vary; also the ordering of events can vary; and even what events or activities are included can vary. There might be entertainment (or there might not), there might be music (or not), there might be sports (or not); there might be any or all of these in different orderings.

Basic SL systems simply are not equipped to deal with relations that cut across state descriptions. This is one reason for the current concern with temporal logic, and formal theories of action involving time and events. [Allen 1981a, 1981b; McDermott 1981; Konlige 1982]. It is conceivable that a good way of adapting temporal logic to planning could help eliminate some of the problems with the fuzziness of some goals. For example, if somehow someone could come up with a set of conditions equivalent to having a picnic, then at some point in time those conditions would have to have been true in the past in order for the goal to have been achieved. It would have to be true at some time that, for example, in the past (i.e. prior to that time) it was true that there existed certain people and that there existed a certain place (satisfying certain conditions e.g. being outside) and that there existed a certain quantity of food, and that some of the food was consumed by some of the people at the place, etc.

The very convolution of the temporal nesting of the specification, however, reveals one part of the difficulty with this sort of approach; the other is that there is no obvious method of incorporating this sort of specification into a search procedure based on means ends analysis and reduction of differences.

Even if tensed specifications of goals that cut across state descriptions can somehow be incorporated into the SL framework, however, there still remains the other source of fuzziness: the fact that fuzzy concepts often do not seem to be definable by any single set of characteristics or criteria. This generality, vagueness, or "fuzziness" of fuzzy concepts, although a liability as far as incorporation into SL frameworks is concerned, is precisely what makes them useful in planning in realistic environments. We

will have more to say about why this is so after we review the concepts of "Strategy vs. Tactics" and "Epistemological Non-determinism" in the next two sections. For now, we will simply note (pending later justification) that a hierarchical planner for a complex domain can achieve greater flexibility by abstracting to the point of fuzziness (i.e. to the point of using fuzzy concepts) with respect to the primitive operators of the domain.

IV. Strategy vs. Tactics.

"Strategy" is an overworked word, with no definite agreement upon meaning, even among military writers [Wylie, J.C. 1967]. Thus it is hardly surprising that an exact differentiation between strategy and tactics is also somewhat problematic. In the interest of a clarity I will begin by explaining how I understand these two terms; and I will follow with some illustrations which should help to make my use of the concepts clear. At that point we can proceed to a discussion of the importance of strategy to planning. Briefly:

> Strategy is concerned with the global aspects of a problem, tactics with local ones.

Often there is no clear dividing line between the two areas; and, when tactics are consonant with strategy, there may be no difference other than that of the relation of the general to the specific: tactics in this case being the implementation of a strategy in a particular case. Perhaps the clearest way to distinguish between the two is to consider cases where the two conflict.

The dictates of tactics tell us how best to accomplish the particular limited goal presently at hand. (In military parlance the proper use of a particular weapon is often referred to as "tactics" [Palmer, Dave Richard 1975]). In terms of the problem reduction paradigm, we can regard tactics as a concern with how best to accomplish a particular subgoal in isolation. Strategy, on the other hand, is concerned with the relations amongst (sub)goals and how these interactions relate to the final objective (or main goal). If we accomplish a particular subgoal at the expense of the main objective then we may have followed the dictates of tactics with respect to

the subgoal and violated the maxims of good strategy, good tactics being sometimes distinct from sound strategy.

If this seems reminiscent of adages about winning the battle and losing the war, that is exactly right. It may be sound tactics under some circumstances to pursue the enemy when he is routed. But, if this means neglecting some important strategical objective, then it is bad strategy. This is the lesson that Howe forgot when, after having driven Washington from New York in 1776, he followed the Continental army into New Jersey, instead of pushing up the virtually defenseless Hudson Valley. (This is assumming, of course, that the correct strategy for the English side was in fact that ascribed to it by many historians viz. the so-called "Line-of-the-Hudson" strategy. [Palmer 1975]). Doing something for immediate advantage and neglecting the global consequences, then, falls under the heading of using good tactics and bad strategy.

One connection between strategy and planning systems should already be clear given the problems (discussed earlier) that SL systems have with subgoal interactions. Recall the first example where, given a goal, ON(A,B) & ON(B,C), the system tries to accomplish ON(A,B) first, interferring with ON(B,C); or else ON(B,C) first, interferring with ON(A,B). Trying to accomplish either subgoal immediately is thus good tactics (with respect to that subgoal), but bad strategy, since it makes accomplishing the main goal, ON(A,B) & ON(B,C) more, instead of less, difficult.

As we saw in our earlier discussion (and there are many more cases in the literature) there have been numerous attempts to imbue SL systems with some way of dealing with strategical matters, at least as far as subgoal interaction is concerned. Unfortunately, these attempts fall short of real

strategy and remain largely in the realm of tactics.

The basic methods of SL systems are antithetical to the use of global strategies. Searching through a tree generated by applications of primitive operators is far removed from the realms of strategy. It is like a general trying to plan a campaign in terms of the particular primitive actions performed by each individual man or horse. The proper concerns of generals are battles, lines of communication, theaters and bases of operation, supply lines, and so on, not the individual motions of particular soldiers, horses or vehicles. [Wilkens 1980] makes a similar point about chess:

> Using actions similar to those used in MYCIN rules would be similar to a chess system where the action part of each rule was to add to the plausibility score of either (1) one or more of the legal moves in the position, or (2) some prespecified object which could lead (through other rules) to recommending a legal move. Such an approach is not satisfactory for the type of chess reasoning needed in the above problems. Concepts at a higher level than legal moves must be used in this reasoning,...

The methods used by SL systems consist in first trying to form a plan out of the low level operators, and then patching up any subgoal interaction problems through techniques like parallel planning, goal regression etc. In as much as the use of strategy can be viewed as a top down process, SL techniques are bottom up. It is true that the use of hierarchical planning is a step in the direction of a top down process. However, as we have already noted, hierarchical systems at present still have an abstraction space no higher than what can be constructed out of the primitive operators of the domain. And, as we can see from Sacerdoti's work, this sort of "hierarchical" planning by itself does not solve the real strategical problem, the subgoal interaction problem; it is still necessary to use Sacerdoti's parallel

techniques or something similar to correct any errors caused by the "hierarchical" search of the space of primitive operators.

Why is strategy important in planning? Why not just do searching in the space of primitive operators as the SL systems do? For the same reasons that apply to people. When a person wants to bake a loaf of bread, for example, he does not (usually) consider the possibility of doing this in the order: (1) put ingredients in oven; (2) bake; (3) remove from oven; (4) mix ingredients; (5) open packages. Nor does he consider any of the other equally absurd orderings which are either impossible or produce no bread or cause some other difficulty. By and large the person will already know a broad strategy for baking which tells him, without really stopping to think about it, that, for example, it is better to mix the ingredients before baking or that it is better to crack the eggs before adding them to the mixture, etc.

The same sort of thing is true in chess. When a typical tree-searching program (Chess 4.5 [Frey 1977]) considers the following position:

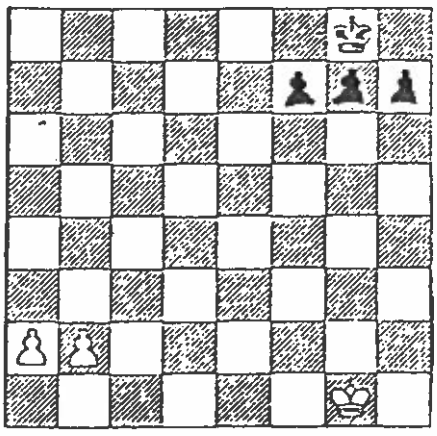

FIGURE 4

(an easy win for White), it must do an eleven ply search before it "sees" that the pawn can queen. What is worse is that if only a nine ply search is performed, 48,273 positions are examined in 96 seconds and the machine then plays the right move, but for completely the wrong reason.[Frey 1977]

Within its horizon of nine ply the program sees the following line:

1. P-R4, P-R4
2. P-R5, P-R5
3. P-R6, P-R6
4. P-R7, P-R7 (check)
5. K x P ...

It chooses this line as best for Black because it is one of the few lines that does not involve the queening of White's pawn within the nine ply look-ahead. (This is because of the interpolated check which "pushes" the queening of the pawn over the horizon [Berliner 1973; Frey 1977]). That is, as far as the program is concerned, this is Black's best line because it prevents the white pawn from queening. Why, then, does it choose 1. P-R4 for White (as opposed to, say, 1. K-N2)? Because in this line, on the fifth move, White captures a pawn and the resultant position gets assigned an extra 100 points by the evaluation function. Thus, the program has no understanding of the strategical implications of the position (all it knows how to do is to search a space of primitive operators i.e. moves). If the position had been slightly different it would have selected completely the wrong move, pursuing the win of a mere pawn at the expense of losing the entire game. (The small

home computer chess games currently on the market look only from three to five ply ahead and often move into a one or two move checkmate while maintaining some local (tactical) function at a maximum).

A human chess player, on the other hand, takes one look at the position in Figure 4 and adopts a well known strategy: when you have a passed pawn, try to win by queening the pawn. The human does not go through an eleven ply search and then stumble with surprize on the move P-R8 (Queen); on the contrary, he starts out with this strategical objective in mind, and organizes his play around it.

If planning systems (and chess programs) had some method of using strategies in a way similar to that of humans, the following seem to be two immediate benefits: (1) many problems with subgoal interactions could be avoided, if the strategies gave either explicit orderings of the subgoals or criteria for ordering them; (2) the "combinatorial quagmire" could be navigated more efficiently, since a predetermined strategy can often automatically exclude many possible operator applications. It remains an open research issue, however, exactly how this might be done. The state of planning in chess reflects this backward state of planning systems with respect to strategy (in chess "planning" is generally used to refer to more strategically oriented play, also called "positional play" [Lasker 1960; Golombek 1964;]).

The most recent attempts at incorporating "planning" into chess playing systems leave much work to be done. [Wilkens 1980; 1982] while employing many of the most sophisticated techniques used in AI (see [Waterman 1978]) only succeeds in devising a system which in Wilkens' own words: "...finds the best move in tactically sharp middle game positions from the games of chess

masters ... The phrase 'tactically sharp' is meant to imply that success can be judged by the gain of material rather than a positional advantage." (Wilkens adds later that the system must also be given a position in which it is winning). [Wilkins 1980] This, of course, takes the abilities of the system right out of the area that in the chess literature is called "planning" (or what we have been referring to as "strategy") and into the realm of tactics.

V. Epistemological Non-determinism

SL systems (e.g. blocks world planners) assume that a sequence of actions can be found which is known (in advance) to define a sequence of states leading from the initial state to some state in which the goal is realized. However, this assumption, which lies at the very heart of the STRIPS paradigm, is justified only for limited and completely deterministic domains. In order to generate the search space (and find a solution path) it is necessary to know what the effects of each action will be. In SL terminology, we must be able to give the add and delete lists for each operator. Unfortunately, in the real world, things are not this simple; many domains exhibit one or more types of what I will refer to as "Epistemological Non-determinism." Not only do we often not know what the effects of an action will be, there is also the problem of not knowing whether the effects produced by the action will persist until the time when the next action in the sequence is to be performed by the system. Thus it becomes impossible to calculate a path through the search space ahead of time.

Ordinarily we speak of a world as being "deterministic" if every effect is inevitably predetermined by its cause(s), and as "non-deterministic" if this is not the case. The problem with the real world is that no one is in a position to know whether determinism holds of it or not. This is true regardless of what in fact is the case, that is to say, regardless of whether determinism actually does, as many believe, hold in the real world.

Even if determinism is true we cannot know it because no one can be aware of every cause and every effect, nor of every connection between them. When we take a walk down a corridor we do not always know ahead of time who we will

meet, or what will be the consequences, any more than we usually know what will be in our mailbox before we look inside. Thus, from an epistemological point of view, that is, as far as the state of our knowledge is concerned, there are many cases in the real world where determinism might just as well be false.

There are many possible sources of this "Epistemological Non-determinism," and a domain need not be isomorphic to the real world to exhibit this property. As a (most likely incomplete) list of things which can give rise to Epistemological Non-determinism consider:

(1) outside agency
(2) combinatorial complexity
(3) non-local effects
(4) intervening factors
(5) effects situation dependent
(6) errors
(7) subgoal interactions
(8) unexplored territory/incomplete information
(9) fuzzy concepts

We discuss each of these below, noting however, that these factors usually do not occur in isolation (i.e. there is much overlap between them). The somewhat artificial division into these headings is only a convenience for purposes of discussion.

(1) Outside Agency.

Almost without exception SL systems assume that there is only one agent capable of producing change in the environment by its actions viz. the execution portion of the planning system (in the case of a robot system, the robot itself). Very little work has been done on the problems involved in multi-agent planning [Carbonell 1979; Konlige 1982]. There are many different ways of viewing these problems, but what is relevant to our present discussion is this: No matter how good the system (or any purposeful agent)

is at predicting the future actions of other agents there is always a chance that the predictions will be wrong. (Often this will result from problems with (9) Incomplete Information, either missing information about the intentions or goals of particular agents, or possibly complete ignorance even of the existence of certain agents). The result is that at any time during the execution of the sequence of actions constituting the plan, some action by another agent can (unpredictably) obtrude and make some action impossible. (Everyone who is married is familiar with this effect). This poses severe problems for an SL system which assumes that relevant states of the world (in particular the preconditions of operators produced by the add lists of previous operator applications) can be known in advance.

(2) Combinatorial Complexity.

Perhaps the best example of this comes from chess. If we formulate the domain description in such a way that the primitive actions are legal moves, the preconditions can be those of legality for the particular piece, and the add and delete lists will have to do with what pieces occupy which squares. We can formulate the goal of checkmate in terms of logical combinations of a predicate ATTACKED(X) which is true when X is attacked by some enemy piece, a predicate NEXTO(X,Y) which is true when X is a square next to square Y, a predicate SQUARE(X) which is true when X is a square on the board, and a function LOCATION(X) which gives the square on which the piece, X, may be found:

```
For all x  (SQUARE(x) @& NEXTO(x,LOCATION(KING)) --->
                ATTACKED(x) @& ATTACKED(LOCATION(KING))).
```

Thus it is theoretically a "simple" problem to find a path through the search space of possible moves to a position where the goal description is true. Fortunately for the game of chess, it is well known that the total number of possible moves (even using heuristics to reduce the number of moves considered) is far too large for this to be computationally feasible. (For the effects of (2) on efforts to do a forward search with an evaluation function see the "Horizon Effect" in [Berliner 1973; Frey 1977]).

(3) Non-local Effects.

The problem here is most easily described in terms of the add and delete lists. In SL systems the effects of an action are completely defined by its add and delete lists, and an assumption is made that the system need not consider any other aspects of the action. This, of course, is a result of a conscious attempt to deal with what is often called the "frame problem." (This problem has nothing to do with Minsky's equally famous "frames"). The frame problem is concerned with the problem of specifying what will change and what will remain the same once an action is performed. The term "frame" might be thought of as an analogy with the frames on a piece of film from a movie. Typically the change between any two consecutive frames of a motion picture is very slight, and most of the contents of the previous frame will be repeated. In many cases it is computationally infeasable to completely specify all the details of what will change and what will not. (See [Hayes 1973]).

SL systems "solve" the frame problem by adopting the assumption that anything not mentioned in the add and delete lists of an operator remains unaffected by the performance of the corresponding action. This assumption works in domains like the blocks world (provided we add that the robot cannot lift more than one block at a time) but as has been noted [Waldinger 1975]

problems arise with this assumption even in as relatively constrained a domain as automatic programming. By "archeological model" Waldinger refers to the use of the Planner context mechanism for world modeling:

> Suppose we attempt to express an assignment statement, X := Y, by updating an archeological model. We must delete any relation of form P(X); furthermore, for every relation of form P(Y) in the model we must add a relation of form P(X). In addition, we may need to delete a relation of form "there is a z such that z has value b" even though it does not mention X explicitly. We may need to examine each relation in the model in order to determine whether it depends on X maintaining its old value. The consequences of this instruction on a model are so drastic and far reaching that we cannot afford to delete all the relations that the statement has made false.

Chess provides another example of an extremely well constrained domain where an action often has effects far beyond what can be included in an add or delete list for a single operator. A single move of a knight or pawn on this side of the board may alter the situation drastically on another remote part of the board (or it may not --- in the case of chess this effect is usually closely linked with (5) Situation Dependence).

(4) Intervening Factors.

This is exactly analogous to (1) Outside Agency, and the import should be obvious. If events outside of those produced by the system can occur and alter the environment, then those events may so alter the environment as to make some particular operator application impossible. This sort of problem does not arise with automatic programming or an artificial domain like the blocks world, but is especially important for any sort of mobile robot which must deal with a possibly volatile environment. Surprisingly little has been done in this area. The basic strategy has been to detect failure during the

execution phase and then return control to the planning phase with the new situation as input (and hope that the goal can still be reached from this new situation). See "Errors" below.

(5) Situation Dependent Effects.

This has already been mentioned in connection with chess under the heading of "non-local effects". The basic idea is that the same action may have different effects in different situations. Thus we cannot simply have an add and delete list which is frozen for all time and all situations.

An example which reveals some of the problems involved is Sacerdoti's ladder case: if we actually let the "apply paint" operator delete "has ladder" as he suggests, then there will be situations where this is not the apropriate way to model that action. On the other hand, if we do not do so, then it is difficult to see how the system can be aware of the possible conflict between its two subgoals.

The worst problem with situation dependent effects, however, is that the effects of actions can depend on aspects of the situation in ways which are not amenable to calculation in advance. Thus even assuming that the system had some way of solving all the other problems, and that it could actually determine in advance exactly what the situation (state description) would be at the time of doing some action, A, the action could be such that its effects could not be calculated from this information alone. It might be that the only way of knowing A's effects would be to actually perform A and wait to see what happens.

At least one researcher is convinced that there is a whole class of actions like this which need somehow to be incorporated into the planning process. [Haas 1982] investigates the problem of planning (as a subgoal) to acquire knowledge needed to pursue another goal. This is extremely important in real domains where the system has incomplete knowledge (see (9) below) and for a robot which uses sensory inputs in its planning. In some cases, where the possible outcomes of the attempt to acquire knowledge are known in advance, it is possible to use "conditional planning" [Wilkens 1980] (the equivalent of a case statement with a suggested path for each possible outcome). But this opens the way to the combinatorial explosion; and, in addition, it is often not possible to anticipate every possible outcome in advance.

(6) Errors

No system can have a perfect correspondence between its representations and the world (with the exception of a system like the blocks world planners where the "world" is artificially constrained to be isomorphic to the system's internal theory of the "world"). The result is (as many of us know from painful experience) that systems make mistakes [Goldkind 1982]. If we lump together mistakes and malfunctions under the general heading of "failure," the effect, as far as Epistemological Non-determinism is concerned, is roughly the same as that of outside agency and intervening factors. (The intervening factors are accidentally caused by the system itself). Some work has been done in this area but the methods are primitive. The basic SL strategy is to divide things into two seperate phases: (a) plan construction, and (b) plan execution. If there is some sort of failure (i.e. if for any reason the state encountered differs significantly from the state predicted) during the

execution phase, control is returned to the planning phase. At this point the planner either attempts to find some way of moving from the present state to one which is part of the predicted sequence, or else begins the costly process of plan construction all over again, taking the present state as the new initial state. Thus the SL planner knows nothing about the possibility of error, and encounters it only as an anomalous condition during the execution phase.

Many researchers see such "anomalous" conditions as part and parcel of non-simple activity in a real world environment, and think that some way should be found to make a planning system's response to such situations more flexible than a simple plan-execute-replan cycle permits. [Srinivas 1978] deals with the problem by having a special module which has access to information about possible causes of failure associated with each type of action. However the method depends on having the appropriate information about possible failures stored ahead of time. More recent suggested solutions are to do away with the seperation into planning versus execution phases [McDermott 1978; Sacerdoti 1979] or to use meta-planning [Wilensky 1981a; 1981b]. Much work remains to be done along these lines.

(7) Subgoal Interactions

We have already addressed the problem of subgoal interactions at some length, and need not repeat any of that discussion here. It suffices to note that one important source of epistemological non-determinism is lack of knowledge about how subgoals acheived will interfere with other subgoals in the future.

(8) Unexplored Territory/Incomplete Information

This may be the source of Epistemological Non-determinism with the greatest practical importance if we are to use mobile robots for the purpose of exploring and charting new territories (underwater, on the moon, other planets, etc...). The usual domains for SL systems are such that the "initial state" can be, and is, completely known. For example, in the blocks world case, the number, names, and locations of all the blocks are known in advance. Thus all possible combinations of operator applications, along with their resultant states can (in principle) be calculated in advance. By definition, this cannot be the case when a robot is sent to explore unknown territory (otherwise there would be no reason to explore it). The effects are much the same when dealing with a complicated domain where there are too many possibly relevant facts to deal with economically (even if they were all knowable in advance), since the planner must defer acquiring specific knowledge until (plan to acquire it at a time when) it becomes necessary to have the information. In either case, the SL system is not equipped to do the appropriate planning when given incomplete knowledge of the initial state.

SUMMARY:

Given the SL assumption that a sequence of actions needs to be calculated in advance of plan-execution, and that the sequence must lead in a deterministic manner from the initial state to the goal state, epistemological non-determinism presents an extremely difficult problem, since the primary effect of epistemological non-determinism is to make it impractical (if not impossible) to construct such a plan ahead of time.

Any one of the sources of epistemological non-determinism discussed above would, by itself, present grave problems for SL systems. Of course it is possible that in each such case ad hoc methods could be found which would provide solutions for simple domains (as in the case of subgoal interaction), but when it is considered that there are many different sources of epistemological non-determinism, it is clear that a more unified approach is desirable.

If planning systems are to move from constrained deterministic domains to complex real world applications involving mobile semi-autonomous sensing robots, then some way needs to be found to give planning systems a way of interacting more flexibly with the environment. It is now clear that the limited methods of SL systems do not suffice [Sacerdoti 1979; McDermott 1978; Wilensky 1981a]. Proposed solutions to the problem involve interleaving planning and execution [Sacerdoti 1979; McDermott 1978], planning to acquire knowledge (i.e. knowledge necessary for further planning) [Haas 1982], and meta-planning [Wilensky 1981a; 1981b]. In the following sections I will suggest a way of unifying these proposals.

VI. Fuzzy Algorithms

We have seen then, that there are several difficulties which planning systems must overcome. Among these are at least (1) combinatorial complexity, (2) subgoal interaction, and (3) Epistemological Non-determinism (all of which may occur and interact in various combinations). Many SL systems have been enhanced in various ways to attempt to deal with (1) and (2), but, I have argued, the very framework of SL systems (in particular the technique of searching a space of world models based on operators composed of add and delete lists) foredooms such attempts. In addition, while some of the sources of Epistemological Non-determinism have been attacked, no one seems to have addressed these problems as a connected whole (except to note their difficulty [Sacerdoti 1979; McDermott 1978]). Yet the problems arising from Epistemological Non-determinism are essentially those that arise in complex, non-static, real world environments.

Assuming that I am (more or less) correct in my assessment of the underlying causes of the difficulties in dealing with such real environments, it is time to suggest directions in which to look for solutions to these problems.

In connection with the chess example of Figure 4 the point was made that rather than searching through a space of legal moves (primitive operators) in a bottom up fashion, the human player could start out with a particular strategy in mind (viz. to win by queening the passed pawn). This, I suggested, is the way that humans deal with the "combinatorial quagmire," and presumably, if we could have a system plan in a top-down manner, then many

problems with the size of the search space could be alleviated. In addition I suggested that using global strategies, instead of searching bottom-up through a space of primitive operators, could provide a mechanism for avoiding much of the difficulty encountered with subgoal interaction problems. This, then, should be one of the main objectives for making progress in planning systems: the development of mechanisms for global planning or strategy.

Whatever the mechanism for acheiving the ability to use strategy (do global planning) it must be able to cope with the problems posed by Epistemological Non-determinism. To do this it must be flexible enough to deal with the uncertainties and incomplete knowledge of real world environments. A rigid, prefabricated sequence of primitive actions which must be executed subsequent to the creation of the sequence of actions will not do; the creation of the plan must interact with the environment as the plan is created (i.e. planning and execution must be interleaved [Sacerdoti 1979; McDermott 1978; Goldkind 1981]).

In this section I will propose the notion of a "Fuzzy Algorithm" as a device for incorporating both strategy and flexibility into a planning system.

Fuzzy Algorithms

We begin by distinguishing between a fuzzy algorithm and a fuzzy statement of a non-fuzzy algorithm. We are all familiar with fuzzy statements of algorithms such as high level descritions (natural language) flow charts, pseudocode, etc. It is characteristic of these descriptions that they do not specify the algorithm completely with respect to every level of detail needed to actually implement the algorithm. But in the case of an ordinary (non-fuzzy) algorithm, it must always be possible to completely specify the algorithm in terms of some set of well defined primitives (for some real or virtual machine) in such a way that any process instantiating the algorithm is guaranteed to produce the outputs specified by the algorithm for given inputs [Knuth 1968]. Moreover, the algorithm can be filled out to this level of detail before being implemented and before any particular inputs are "given." A fuzzy algorithm differs from this in that it is impossible to specify every detail of the algorithm in advance of execution, and also in that there is no guarantee that the implementation and execution of the fuzzy algorithm will produce the desired results.

Consider the way an ordinary algorithm might be instantiated (implemented). We might start with a vague or fuzzy specification at a high level, and then go through a series of transformations between different levels, for example:

high level description (e.g. natural language)
pseudocode
high level language
intermediate code
machine level (virtual or real).

In going from any one level to the next there is no one particular transformation that is the "right" one, but there is at least one that is guaranteed to produce a sequence of operations at the next lower level which, on the right virtual machine, will produce the input output pairs of the algorithm.

In the case of a fuzzy algorithm, not only is there no one "right" transformation, but in addition, we don't even know ahead of time that there is even one transformation that is guaranteed to work. This is because a fuzzy algorithm has to deal with objects external to the algorithm which may behave in unexpected, possibly even disorderly ways [McDermott 1978]. Thus, depending on sometimes unpredictable changes in these external objects, it may be necessary to overcome additional unforseen obstacles, do things in a different order, in short: to respond flexibly to the environment. Again, the main point is that the system "following" a fuzzy algorithm is concerned with objects which are <u>not</u> simply internal data structures which are reasonably well behaved, but real objects in the world which are subject to influences the system may not even be aware of [Goldkind 1982]. (Actually we have already seen that even internal objects of a certain complexity can cause problems [Waldinger 1975]).

What has been said up to this point about fuzzy algorithms (henceforth "FA" for singular or plural) was not supposed to be a complete account but, rather, a broad overview meant to provoke in the reader such questions as: (1) Is there really such a thing as a FA? (2) Even if there is such a thing as a FA, and even if people somehow use them, (a) how can they be specified and (b) even if they could be specified, how could a machine "follow" such a

specification?

I hope to skirt the issue of (1) by answering (2) (since if we can specify FA, and machines can use them, then we have some evidence for their existence). At the same time I will try to answer what was our original question: (3) How do FA contribute to strategy and flexibility? In answering (2), however, I do not mean to imply that a particular representational formalism has already been found; on the contrary, that is a matter for further research. The examples given serve merely to give some idea of how one might go about representing FA. (Any resemblance between the representation given here and some future formalism is likely to be coincidental).

Plan Schemas

To be explicated: "A FA consists of a hierarchy of fuzzy plan schemas which is determined by the heuristics for instantiation accompanying each schema in the hierarchy." We begin by explaining the notion of a fuzzy plan schema (FPS for singular and plural).

Even the most specific sequence of actions prefabricated by an SL planner has a certain amount of generality to it. This is due to an interplay between the specification language and the world; any one state description has numerous possible instantiations (realizations) which may differ in ways not captured by the specification language. The basic idea behind hierarchical planning, as we have seen, is to allow more generality than this in the "higher" levels of planning by ommitting details that are capturable by the

specification language, but which are relatively less important. The motivation behind the idea of a FPS is to put even more generality than this into the planning process.

A plan schema is something which (if we adhere to the by now almost "traditional" view of a plan as a sequence of actions) determines not just one particular sequence of actions, but rather, a set of sequences of actions each of which is a possible instantiation of the plan-schema. This is consonant with the idea of having the FA incorporate strategy into the planning process. Strategy need not determine an exact sequence of primitive actions which will be used to carry out the plan (although in simple cases it might). It serves instead to constrain the primitive actions in various ways, the exact particular actions being a matter of tactics, not strategy. As a simple example of a plan-schema consider:

```
(GOAL Playchess
   (SUBGOALS  (1 Playopening)
              (2 Playmiddle)
              (3 Playend)))
```

A plan-schema, then, must give some kind of description of a broad strategy that need not contain any mention of the particular actions which instantiate it. And planning involves the process of instantiating the general strategy described in the plan-schema. Notice that if the plan-schema were to specify subgoals which could in turn be specified completely we would have been returned rather quickly to a standard sort of problem reduction, say for example:

```
(GOAL Give-intersect (X Y)
   (SUBGOALS (1 (All x IN X (CONSTRAINT (x in Y)))))).
```

What is required is that some or all of the subgoals referred to by a plan-schema be fuzzy concepts (as described in Section IV). When this is the case we will say that we have a FPS, and to make things easier to talk about, we will consider ordinary plan-schemas to be simply degenerate or vacuous cases of FPS.

Several advantages of the use of FPS are immediately apparant. First of all, we are not limited to primitive operators at our higher levels of abstraction. In addition we are not limited to decomposition by logical connectives as the only form of problem decomposition. And, of course, having a predetermined strategy can often reduce the need to search through the space of primitive operators.

So far all of this is fairly straightforward (except for a description of how instantiation of the FPS is to occur, to which we shall return shortly) and could be accommodated by minor alterations to existent theories. A Script, for example, could be construed as a plan-schema (see also the notions of plans and goals presented in [Shank and Abelson 1977]). Scripts were developed in connection with natural language understanding, and it might seem coincidental that Scripts could offer the possibility of serving as plan-schemas. However, the connections between story understanding (and hence natural language understanding) and planning are now widely recognized [Bruce and Newman 1978; Wilensky 1981a; 1981b; 1983].

In order to understand a story, a system must be able to understand the actions and goals (plans) of the characters in the story. If we see plan-schemas as a way of allowing the planning system to manipulate general schemas for plans, and in a sense use a schema to "understand" a sequence of

actions, then it does not come as a great surprise that a concept used in story understanding has potential utility for planning systems. In fact, as least one researcher is attempting an integrated system which performs both story understanding and planning [Wilensky 1981a; 1983].

However, given that a planning system is to employ FPS, and that FPS contain fuzzy concepts (for flexibility and generality) which indicate the subgoals associated with a particular goal, we come to a seemingly difficult problem: if the fuzzy concepts are to be used as subgoals, and fuzzy goals, by hypothesis, cannot be easily defined or specified, what good does it do the system to have a FPS? Admittedly it is fine to give a division of the goal of playing chess into the further problems of playing the opening, middle, and endgame; but if playing the opening is to be explained by a fuzzy concept, what good does this do? How is the system to get from the vague information that it is supposed to play the opening, to the point of actually choosing a particular move to play?

Part of the answer clearly must be that somewhere the system has information about each of the subgoals in the FPS. Since each subgoal is a fuzzy goal which cannot be completely specified yet needs to be somehow specified in order to be used by the system, we choose to use another FPS to specify it, for example:

```
(GOAL Playopening
   (SUBGOALS (1 Playsicilian)
             (2 Playdutch)
             (3 PlayQP)
             (4 PlayKP) ... [etc.])).
```

And in order that each of these further subgoals can in turn be a fuzzy concept, each of these are also specified by a FPS. And etc. Then, in order

that the reduction into subgoals terminate at some point, we must have ordinary plan-schemas (and perhaps even individual actions) or "degenerate" FPS somewhere at the bottom of this hierarchy of FPS.

This of course, presents us again with a problem: if a partial ordering can be traced through this hierarchy of plan-schemas which terminates at leaf-nodes which are non-fuzzy (i.e. determinate or completely specifiable) then there is no reason to call the various subgoals throughout the hierarchy "fuzzy goals" (since they can, by a process of reduction, be completely specified). On the other hand, we are faced with an infinite proliferation of subgoals if the process does not terminate somewhere with something definite.

The answer lies in the following addition to the basic idea of a plan schema: A FPS consists of a plan-schema (whose subgoals are fuzzy goals) plus heuristics for instantiation. For example:

```
(GOAL Playchess
   (SUBGOALS  (1 Playopening)
              (2 Playmiddle)
              (3 Playend))
   (HEURISTICS  (a (SEQUENCE 1 2 3)) ...)...)
```

Here we have added one heuristic for instantiation, namely an ordering requirement on the subgoals which states that the sequence of instantiation is always to be <Playopening, Playmiddle, Playend>. Of course, this particular FPS differs not at all in effect from the previous version of Playchess as far as incorporating any "fuzziness" or flexibility: there is nothing situation dependent in this specification, and the order of subgoals is predetermined for all cases. But the intention is to allow the use of heuristics which are situation dependent to determine (among other aspects of instantiation) orderings on subgoals. (For example, if the system were not always required

to play a game straight through, then there might be heuristics that it would apply to the board position to determine which phase of the game it was in).

By allowing arbitrary subgoal orderings to be specified we accomplish part of the goal of allowing for strategy with regard to subgoal interactions, the heuristics embodying much of the strategy component. The orderings can of course in principle be much more complicated, for example:

```
(SEQUENCE 1 {7 5 6} {4 3} 2 8 9)  or

(SEQUENCE 1 (COND [(ANY {7 5 6})(3 4)]
                  [T (2 4 {7 6} 5)]))
```

where set braces, '{' and '}' indicate independent subgoals to be acheived in any order (or concurrently) and 'COND' has its usual meaning. Also, the possibility exists of having the sequence specified less directly in terms of other FPS.

This ability to deal with subgoal orderings is an important capability, but the most important source of fuzziness and flexibility FPS lies in the ability to specify arbitrary heuristics. For example, consider the problem of deciding, (for the Playchess FPS) when it is time to move from the Playopening phase to the Playmiddle phase. There is no set number of moves which marks the boundary between opening and middlegame, nor is there any rigid set of criteria which is always right. But we might expand the Playchess FPS thus:

```
(GOAL Playchess
   (SUBGOALS (1 Playopening)
             (2 Playmiddle)
             (3 Playend))
   (HEURISTICS  (a (SEQUENCE 1 2 3))
                (b (TRANSITION-INFORMATION 1 2
                    (MESSAGE TO 1
```

```
["WHEN condition (MESSAGE TO here
 ["condition-found"]"])
(WHEN (READMESSAGE = condition-found)
   ... further heuristics ...followed by
   either (DO 2) or (CONTINUE 1)
       ...)...)..)).
```

No claim is being made here as to the correctness of this way of taking care of the transition, or that these lisp-like formulations are the best way to represent FPS. Clearly some sort of message passing facility is envisioned, along with the ability of processes to create (sub)processes and monitor them. In this case, the method is for the controlling process (designated in the FPS above as "here") to post a message which indicates to the process instantiating Playmiddle some condition which signals the possibility that a transition may be required. Here we only show one condition which is to trigger a demon-like response, but it is likely that a sequence of conditions would need to be considered, with the addition of each satisfied condition increasing the probability that a transition is actually required (and possibly also requiring that additional heuristics or even FPS to be used). This sort of incremental interaction between processes instantiating different levels of FPS (in the terminology of hierarchical planning we would say different "abstraction spaces") should contribute to the ability of a planner to deal in flexible ways with its environment. (See [Hayes-Roth 1979a] for a description of exactly this sort of flexibility which they call "heterarchical planning"). We note again that exactly what the primitives should be for the representation of FPS, and exactly what the mechanisms for instantiation should be, are subjects for further investigation. For now, we observe that the answer to the question of 2(a) is:

FA are specified by a hierarchy of FPS, the hierarchy being determined by the heuristics in each of the FPS.

But what about 2(b)? How is the system to "follow" or "execute" a FA? The answer is that this is done incrementally, moving along a sequence of FPS as and when the heuristics select the next member of the sequence from the hierarchy. (This should allow the system, as a special case, to simulate the so called "blackboard" approach [Hayes-Roth 1979(a-c); Feigenbaum 1983]). This sort of incremental approach allows for greater flexibility [McDermott 1978]. This is not an entirely new idea; several researchers have suggested using hierarchical techniques and leaving various parameters un-instantiated as long as possible [Sacerdoti 1973; Stefik 1981a], but the idea here is to incorporate even more flexibility than this.

As a result of Epistemological Non-determinism, it is often necessary to completely abandon a plan and try something completely different. For example, in chess, it often happens that the opponent makes a bad move which renders one's whole plan superflous or even inapplicable. As the system moves along the sequence of FPS it is called upon by heuristics to execute primitive actions (degenerate FPS) but it need not always know ahead of time exactly what it will be doing. For example, at the top level, if it is currently playing the middle game, it may know that eventually it will do "Playend", but until it gets to that point, it may not need to do any calculations involving any of the particulars of exactly how this will be accomplished. Thus the cost of changing tactics within any one subgoal need have little or no effect on the overall strategy at a higher level. In addition, and this is the improvement over most hierarchical schemes, when the system needs to adopt some new tactics (or even a completely new strategy) all that is required is

that it hook up with some appropriate FPS. (We will note later that this may be done through the use of meta-planning). Once it has done so, a new FA is entered upon based on the heuristics associated with that FPS. Errors, failures, or anomolous conditions can be handled as special cases of changing plans and, in addition, there is available the possibility of including specialized heuristics in a FPS to deal (in a more ad hoc way) with the failures of specific subgoals (in cases where these are foreseen).

VII. Meta-Planning

The importance of meta-planning is widely recognized and we will not attempt a comprehensive survey of the area here. (See [Wilensky 1980, 1981a, 1981b, 1983; Stefik 1981b; Haas 1982; Hayes-Roth 1979; McDermott 1978]). Some aspects of meta-planning are already clearly capturable by FPS given the description of the last section; others require further explanation. We mark the latter cases with a '*'.

[Wilensky 1981a] notes several important features of meta-planning:

*(1) Everyday planning situations often involve various goals that can interact in complicated ways. If both A and B are subgoals of one particular FPS, P, then it is possible (as mentioned previously) for the heuristics of P to mediate potential interactions or conflicts between A and B. The interesting case, however, is where this is not the case, so that A and B are either completely independent (except for the overlap which produces the interaction) or else the inter-relation of A and B is not forseen prior to actually detecting the interaction during planning. In this case there will be no one particular FPS with heuristics that already deal specifically with the combination of A and B, but a planning system should be flexible enough to deal with cases like this. The solution which is advocated by Wilensky and others is to allow the system to deal with the problem of the interaction (whatever it might be: e.g. avoiding goal annihilation or combining two similar activities to gain efficiency) as a new planning problem. To deal with the problem of how to take care of the interaction, the system can have "knowledge about planning" (in order to decide how to plan to take care of the

interaction properly). This is what Wilensky refers to as meta-planning. The advantage is that if these meta-problems can be formulated as goals (albeit meta-goals) they can be solved using the same general methods that the system normally uses (without, for example, having to consult special purpose critics).

The notion of FA accomodates this idea easily, since all that is required is that there be FPS for dealing with various types of interactions (or any other meta-problems). That is, we should include in our store of FPS ones which deal with planning as subject matter.

(2) In most planning systems, high-level goals are simply handed to the planner, often in the form of a problem to be solved or a state to be reached. But a semi-autonomous planner (e.g. an exploratory robot out of contact with its home base) must be able to infer its goals based on its overall mission together with the situation in which it finds itself. This can be accomplished by FPS as described in the last section. Wilensky's example is of a system for maintaining a nuclear reactor. The robot is in charge of sustaining the generation of power, keeping the floors clean, preventing meltdowns, cleaning up dangerous spills, and maintaining itself. But usually most of these goals should have no effect. For example, the robot is not concerned with cleaning up a spill until one occurs.

It seems desirable then, to design a planner which is capable of recognizing situations in which these tasks should be performed. This could be accomplished with a single FPS having each of these as subgoals and heuristics for determining which subgoal to execute next (i.e. in a sense FPS are already "meta" plans with respect to their subgoals).

*(3) Meta-planning knowledge can be used for both planning and understanding. Wilensky's version of meta-planning requires that something be represented declaratively in order to qualify as meta-planning knowledge. Although this is somewhat narrow if construed as a definition (since there is no reason in principle to exclude the procedural use of meta-knowledge about planning) his point is well taken. For natural language understanders to cope with many uses of language they must be able to understand the goals and intentions both of speakers and of agents spoken about. If the knowledge about how to resolve conflicts between goals (or any other meta-planning knowledge) is embedded somewhere inside the control structure of the system, then it is practically impossible for the system to utilize this knowledge in explaining, understanding, or reasoning about the goals and behavior of other systems.

FPS, while not tied to a particular representational scheme (at this point) were conceived as a generalization and elaboration of declarative Script-like representations (e.g. [London 1978b]). Thus, the FPS can be seen as declaratively representable. It remains a question for further investigation if and how well the heuristics for instantiation can be captured in a declarative representation; this however, is the intention of the present writer. That is, the first stages of investigation should involve a search for such a declarative representation for heuristics which satisfies the other constraints on FPS (e.g. that it can serve as a meta-language for itself -- see [Perlis 1980; Haas 1982] for some initial progress along these lines).

(4) Meta-planning knowledge allows for more flexibility in dealing with cases where no solution can be found. This is very similar to (1) if instead of of unforseen interactions we consider a complete failure of some sort. Wilensky's point is that if meta-knowledge is used in the first place in the formulation of the goals which failed, then it can be used again to reformulate the goals or to try to plan around the problem. This is not possible if the system has no knowledge about planning (since then it can have no knowledge about planning failures). FPS can acheive this same effect in exactly the same way as suggested in (1) viz. have FPS for error recovery. This can be done in an ad hoc way within particular FPS for particular applications and also by having Meta-FPS which deal with more general questions. (See [McDermott 1978; Wilensky 1981a] on "themes" and how they give rise to meta-goals).

Aside from these important effects of meta-planning mentioned by Wilensky and others, there are some additionl benifits to be acheived by a particular sort of meta-planning within the framework of FPS:

*(5) Meta-planning can be useful for knowledge acquisition and learning (see [Davis 1982]). If FPS can deal with other FPS as objects (i.e. from a meta-level) then a system can have various (meta)FPS which tell it how to acquire, construct, delete, alter, and manipulate FPS. Some work has been done along these lines by [Davis 1982], and this is an important area of current research, since one of the bottlenecks for expert and knowledge based systems is the problem of how to get the "knowledge" into the system. Related research is underway in the area of intelligent interfaces to database systems. The idea which ties these areas together is that of a database

interface which has (meta)knowledge of the data base schemas and can interrogate the user to obtain data and construct instances of the schemas. In the case of a planning system using FPS, however, (since a FPS is a more complicated sort of schema than most data bases handle) there is a possibility of going beyond the basic idea of acquiring knowledge about objects and fitting the knowledge into existant schema. There is also the more exciting possibility of altering existing schema and acquiring new ones (including Meta-FPS). This has been explored to some extent by [Davis 1982] but FPS offer the additional possibility of changing the schema for acquiring schema. That is, there exists the possibility of having every activity (with the exception of some small kernal) governed by the use of appropriate FPS, including the activity of acquiring new schema and altering old schema. If arbitrary schemas can be altered, then even the schemas controlling schema change can be so altered. Thus there seem to be possibilities for research into the abilities of this sort of system to learn.

Allen, J. F.: "Maintaining Knowledge About Temporal Intervals," TR 86, Department of Computer Science, University of Rochester, Rochester, NY 1981a.

---------- "A General Model of Action and Time," TR 97, Department of Computer Science, University of Rochester, Rochester, NY 1981b.

Berliner, Hans J.: "Some Necessary Conditions For a Master Chess Program," IJCAI-3, August 1973.

Bruce, Bertram and Newman, Denis: "Interacting Plans," Cognitive Science 2, 1978.

Carbonell, J. G.: "The Counterplanning Process: A Model of Decision-Making in Adverse Situations," Rep. Computer Science Department, Carnegie-Mellon University 1979.

Corkill, Daniel D.: "Hierarchical Planning in a Distributed Environment," IJCAI-79, 168-175, August 1979.

Davis, R. and Lenat, D.: Knowledge-Based Systems in Artificial Intelligence, McGraw-Hill, New York, 1982.

Ernst, G. and Newell, A.: GPS: A Case Study in Generality and Problem Solving, ACM Monograph Series, Academic Press, New York, NY 1969.

Feigenbaum, E.: Lecture at Computer Science Department, Washington University St. Louis, Missouri 1983.

Fikes, Richard E., Nilsson, Nils J.: "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," Artificial Intelligence 2 (1971), 189-208.

Fikes, Richard E.: "Monitored Execution of Robot Plans Produced by STRIPS," Proc. IFIP Congress 71, Ljubljana, Yugoslavia (August 23-28, 1971).

Fikes, Richard E., Hart, Peter E., and Nilsson, Nils J.: "Learning and Executing Generalized Robot Plans," Artificial Intelligence 3, 1972.

Frey, Peter W.: Chess Skill in Man and Machine, Springer-Verlag, New York, 1977.

Goldkind, Stuart: The Abilities of Machines, Doctoral Thesis, Department of Philosophy, University of Rochester, Rochester, NY 1981.

---------- "Machines and Mistakes," Ratio, v. XXIV No. 2, Basil Blackwell, Oxford England, 1982.

Golombek, H.: "Planning in the Middle Game," in The Art of the Middle Game, (eds) Keres, Paul, and Kotov, Alexander, Penguin Books Inc., Baltimore, Maryland 1964.

Haas, Andrew: Planning and Mental Actions, Doctoral Thesis and TR 106, Department of Computer Science, University of Rochester, Rochester, NY 1982.

Hayes, Patrick: "The Frame Problem and Related Problems in Artificial Intelligence," in Artificial and Human Thinking, edited by Elithorn, A., and Jones, D. San Francisco, Jossey-Bass 1973.

Hayes-Roth, Barbara & Hayes-Roth, Frederick: "A Cognitive Model Of Planning," Cognitive Science 3(4), 275-310, 1979a.

Hayes-Roth, Barbara & Fredrick; Rosenschein, Stan; and Cammarata, Stephanie: "Modeling Planning As an Incremental, Opportunistic Process," The Rand Corporation, Santa Monica, California, Februray 1979b.

---------- "Modeling Planning As an Incremental, Opportunistic Process," IJCAI-79, 375-383, 1979c.

Knuth, Donald E.: The Art of Computer Programming v. 1, Addison-Wesley 1968.

Konolige, Kurt: "A First-Order Formalisation of Knowledge and Action For a Multi-Agent Planning System," Machine Intelligence 10, 1982.

Lasker, Emanuel: Lasker's Manual of Chess, Dover Publications, New York, NY 1960.

London, Philip E.: "A Dependency-Based Modelling Mechanism for Problem Solving," Computer Science Department, University of Maryland (College Park), TR-589, NSG-7253, November 1977.

---------- "Approaches to Object Selection for General Problem Solvers," CS Department, University of Maryland (College Park), TR-632, NSG-7253, January 1978a.

---------- "Dependency Networks as a Representation for Modelling in General Problem Solvers," CS Department, University of Maryland (College Park), Ph.D. Thesis and TR-698, NSG-7253, September 1978b.

Manna, Zohar and Waldinger, Richard: Studies in Automatic Programming Logic, Elsevier North Holland, NY 1974.

McDermott, Drew: "Planning and Acting," Cognitive Science 2, 71-109 (1978).

---------- "A Temporal Logic for Reasoning About Processes and Plans," RR 196, Computer Science Department, Yale University, 1981.

Palmer, Dave Richard: The Way of The Fox, American Strategy in the War for America, 1775-1783, Greenwood Press, Westport Connecticut 1975.

Perlis, Donald: Truth, Syntax and Reason, Unpublished Doctoral Thesis, Department of Computer Science, University of Rochester, Rochester, NY 1980.

Rieger, Chuck and London, Phillip: "Subgoal Protection and Unravelling During Plan Synthesis," TR-512, Department of Computer Science, University of Maryland, College Park Maryland, 1977.

Sacerdoti, Earl D.: "Planning in a Hierarchy of Abstraction Spaces," Artificial Intelligence Center, Technical Note 78, SRI Project 1530, June 1973.

---------- "The Nonlinear Nature of Plans," Artificial Intelligence Group, Technical Note 101, SRI Project 3805-2, January 1975.

---------- A Structure For Plans and Behavior, Elsevier North-Holland, New York, 1977.

---------- "Problem Solving Tactics," IJCAI-79, 1077-1085, August 1979.

Schank, R. C. and Abelson, R. P.: Scripts, Plans, Goals and Understanding, Hillsdale, N. J., Lawrence Erlbaum Assoc. 1977.

Stefik, Mark: "Planning with Constraints (MOLGEN: Part 1)," Artificial Intelligence, 16(2) 1981a, 112-139.

---------- "Planning and Meta-Planning (MOLGEN: Part 2)," Artificial Intelligence 16(2) 1981b 141-170.

Sussman, G. J.: A Computer Model of Skill Acquisition, Elsevier Horth Holland, NY 1975.

Waldinger, Richard: "Achieving Several Goals Simultaneously," Artificial Intelligence Center, Technical Note 107, SRI Project 2245, July 1975.

Waterman, D. A. and Hayes-Roth, Frederick (eds): Pattern-Directed Inference Systems, Academic Press, NY 1978.

Wilensky, Robert: "Understanding Complex Situations," IJCAI-79, 954-959, August 1979.

---------- "Meta-Planning," First NCAI, 334-336, August 1980.

----------"Meta-Planning: Representing and Using Knowledge About Planning in Problem Solving and Natural Language Understanding," Cognitive Science 5, 197-233, 1981a.

---------- "A Model For Planning in Complex Situations," Electronics Research Laboratory, College of Engineering, UC Berkeley, Memorandum No. UCB/ERL M81/49, 1981b.

---------- Planning and Understanding, Addison-Wesley Inc., Reading Massachusetts, 1983.

Wilkens, David: "Using Patterns and Plans in Chess," Artificial Intelligence 14(2) 1980.

---------- "Using Knowledge to Control Tree Searching," Artificial Intelligence 18 1982.

Wylie, J. C.: Military Strategy: A General Theory of Power Control, Rutgers University Press, New Brunswick, NJ 1967.