Case-Based Planning:  
A Framework for Planning from Experience  

KRISTIAN J. HAMMOND  
The University of Chicago  

This article presents a view of planning as a task supported by a dynamic memory. This view attempts to integrate models of memory, learning, and planning into a single system that learns about planning by creating new plans and analyzing how they interact with the world. We call this view of planning case-based planning.  

A case-based planner makes use of its own past experience in developing new plans. It relies on its memory of observed effects, rather than a set of causal rules, to create and modify new plans. Memories of past successes are accessed and modified to create new plans. Memories of past failures are used to warn the planner of impending problems, and memories of past repairs are called upon to tell the planner how to deal with them. 

This view of planning from experience supports and is supported by a learning system that incorporates new experiences into the planner's episodic memory. This learning algorithm gains from the planner's failures as well as its successes. Successful plans are stored in memory, indexed by the goals they satisfy and the problems they avoid. Failures are also stored and indexed by the features in the world that predict them. By storing failures as well as successes, the planner is able to anticipate and avoid future plan failures.  

Case-based planning is aimed at improving planning behavior in three areas: failure avoidance, plan repair, and plan reuse. It also attempts gains over current learning systems, in that the learning is driven by the functional needs of a planner.
1. CASE-BASED PLANNING

People plan from memory. When a surgeon approaches an operation, he does not build his plan step-by-step out of a set of primitive actions. Instead, he recalls past operations performed in similar situations and adapts his previous approach to suit the new situation. When an architect starts a new design for a client, he does not go back to first principles and try all possible combinations of subplans, but rather recalls previous designs and modifies them to fit his current needs. And when you get into your car to go home tonight, you will not reinvent the plan for buckling up, starting the car, and finding a route home, you will just recall the plan that has worked before and make use of it directly.

This article outlines an approach to this kind of planning from memory, called case-based planning.

Case-based planning is the idea of planning as remembering.

Planning from cases means remembering successes so that they can be reused, remembering failures so that they can be avoided, and remembering repairs so that they can be reapplied. Case-based planning differs from rule-based planning in that it rests on the notion that new plans should be based on the planner’s knowledge of what has succeeded and failed in the past. Questions of memory organization, indexing, and plan modification are important in case-based planning because a case-based planner makes extensive use of memory. Learning is central to case-based planning because a case-based planner must reuse, and learn from its own experiences in order to build new plans and to avoid past errors.

Planning is a memory task. A plan for a set of goals is not built up piece-by-piece from the individual plans for each goal. It is, instead, constructed by modifying a plan from memory that already satisfies or partially satisfies many, if not all, of the planner’s goals. A planner’s failures are not only plan failures, they are also expectation failures, which have to be remembered, so that the faulty expectations can be changed. They are indications that the planner’s understanding of the world is faulty and should be altered in much the same way as the plan is altered. Also, plans are not disposable items that should be built and then discarded. They are valuable commodities that can be stored and recalled for later use. The problem of building and maintaining plans is a problem of the interaction between a planner’s knowledge base and the world. Any problem that arises out of a disparity between what the planner knows and how the world is actually structured indicates that the model of the world, a model shaped by the planner’s memories, should be changed.

A case-based planner uses its knowledge of the world and the effects of its actions in the world to build a plan. As a result, the plan becomes a test
of that knowledge. If the plan is faulty, something about the planner’s knowledge is faulty. Plans are recalled from memory and allowed to interact with the world. The feedback from this interaction is then used to modify existing plans and add new plans to memory. These modifications and additions change the planner’s understanding of what can happen in different situations and make it possible for the planner to understand which plans are appropriate for certain situations.

A case-based planner makes use of memory whenever possible. It begins planning by using its memory of past failures to warn of problems. It then searches its memory of successes for a plan that can be modified to fit the goals it needs to satisfy and problems it seeks to avoid. If it has a plan failure, the planner treats it as a failure of its understanding of the world and explains the failure so that it can repair the faulty plan and the knowledge of the world that allowed it to create that plan. Finally, it saves its successes in memory, indexed by the goals they satisfy and the problems they avoid, so that they can be used again to satisfy similar goals and avoid similar problems.

By definition, case-based planners must learn. They must learn new plans in order to store the plans they create for later use. They must learn to predict problems based on their own experiences so that they can find the plans that avoid those problems. And they must learn specific repairs to planning problems that can be applied again when similar problems arise.

A case-based planner is motivated to learn only when that learning will help the planner in later situations. It stores new plans in memory so that it can use them again in similar situations. It learns to associate goals with the problems it encounters while planning for them so it can anticipate and avoid those problems in the future. And it learns new repairs for problems so that it can apply plans with less effort once the problems have been predicted.

2. A NEW THEORY OF PLANNING

If one considers planning problems as memory problems, one is forced to reconsider basic features of most theories of planning:

1. Rather than planning for individual goals and then merging the results, a case-based planner searches its memory for plans that satisfy many of its goals at once.
2. Rather than recovering from planning errors and then forgetting the results of that recovery, a case-based planner treats these errors as opportunities to learn more about its domain and the problems that arise in it.
3. Rather than discarding the plans that it builds, a case-based planner saves them in memory for later use in similar circumstances.
The theory of case-based planning sees planning as an activity that tests a planner's understanding of the world. Planning and learning form a closed loop, in which planning errors lead the planner to learn more about what causes them, which gives it a better understanding of how to avoid them.

The issues involved in case-based planning are not confined to planning in the strictest sense of the word. Although plan modification and validation, which are planning issues in the narrower sense, play a role in case-based planning, the issues of memory, indexing, and learning are far more basic.

In order to plan from past experience, a planner must have a rich understanding of that experience and a clear method for organizing it and incorporating it into memory. The memory structures and learning mechanisms needed to support a case-based planner must have the ability to integrate past failures and successes into memory so that the former can be avoided and the latter reused.

Case-based planning differs from other approaches to planning and problem solving (e.g., Fikes & Nilsson, 1971; Friedland & Iwasaki, 1985; Newell & Simon, 1972; Sacerdoti, 1975; Sussman, 1975; Tate, 1977; Wilensky, 1980) in three areas: in initial plan building, in the reaction to plan failures, and in the vocabulary for describing and storing plans. Although there is a great deal of overlap in these areas—given that the initial choice of a plan affects the way in which it is debugged and that the way in which debugged plans are stored affects the way in which they are chosen for later use—it is important to separate them and understand how different planners handle them.

2.1 Building an Initial Plan
A case-based planner builds new plans out of old plans. It stores these past planning experiences in an episodic memory organized by two sorts of indices: goals to be satisfied and failures to be avoided. Plans are organized around goals so that they can be retrieved when the goals they satisfy are requested. They are also organized around the failures the planner encountered when originally putting them together. Anticipated failures can then be avoided by finding plans that were constructed to deal with similar failures in the past.

In order to use its own memory organization, a case-based planner must begin the task of building a plan for a set of goals by considering how they will interact. By doing this, it can anticipate any failure it has experienced before and use this anticipation to search for a plan to solve the problem it has predicted. It can also use the prediction of any positive interaction between goals to characterize its current set of goals in terms of an already existing exemplar in memory that can be accessed directly. It has to anticipate problems that will arise in order to find plans that will avoid them. This just means that a case-based planner has to be able to infer from the fact
that it is raining that it will get wet if it plans to go outside, and thus, it must find a plan avoiding that problem instead of building and having to repair a faulty plan that does not.

The case-based approach to finding an initial plan is to anticipate problems so the planner can find plans to avoid them.

By organizing plans around planning failures as well as goals, a planner can avoid problems it has encountered before. The planner can use the prediction of a failure resulting from a goal interaction to find a plan that avoids it. This idea of using a prediction mechanism, along with a memory organization able to make use of these predictions to anticipate and avoid planning problems, contrasts strongly with the create and debug paradigm that has been the thrust of machine planning over the past 15 years (e.g., Fikes & Nilsson, 1971; Sacerdoti, 1975; Sussman, 1975; Wilensky, 1980, 1983). The main difference between these approaches is that the anticipate and avoid approach tries to predict problems and then avoid them by finding plans in memory that deal with them, whereas the create and debug approach debugs failures only after they arise during the planning process.

2.2 Debugging Failed Plans
A case-based planner can anticipate and thus avoid failures having to do with plan interactions. It can only do this, however, with interactions it has seen before. In order to plan effectively, it must be able to recover and learn from failures it has not seen before and is not able to anticipate. So, like create and debug planners, it has to have knowledge of how to identify and repair faulty plans that have failed due to unforeseen interactions between steps.

Although there are technical differences between the way plan failures are handled by a case-based planner and the way programs such as NOAH (Sacerdoti, 1974, 1975) or PANDORA (Wilensky, 1980) deal with them, the most important difference between them is that a case-based planner treats its mistakes as expectation failures as well as planning failures. Planning is a test of understanding the world. Planning failures indicate where that understanding has broken down and where it has to be fixed. They tell the planner when it needs to learn.

A planning failure occurs when a plan does not satisfy some goal it was designed to deal with. For example, if a planner puts together a plan to get a newspaper on a rainy day that is a simple "go outside, get paper, come back" plan, it will end up getting wet. Because the planner is wet and doesn't want to be, it has had a planning failure. But because it did not expect to get wet, it has also had an expectation failure. An expectation failure is different from a planning failure. It occurs when an expected event does not take
place or when an unexpected event occurs. In the newspaper situation, the expectation failure occurs at the same time as the planning failure, but the response to the planning failure is the alteration of a plan, whereas the response to the expectation failure has to be the alteration of the planner and its understanding of the world.

A planner must respond to a planning failure by building a causal explanation of why the failure has occurred and then using that explanation to access replanning strategies designed for the situation in general. It must respond to an expectation failure by again using that explanation to add new inference rules that will allow it to anticipate the problem it was previously unable to foresee. It should first ask itself, "What went wrong with the plan?" and then ask "What went wrong with the planning?"

In other words, a planner has to repair its expectations about the world when those expectations lead to plans that fail.

Knowledge repair is one capability which sets case-based planning apart from other theories of planning. The experiences a planner has while planning are tests of its knowledge of the world. Any failure of a plan is a failure that forces a reevaluation of that knowledge.

A case-based planner responds to planning failures by repairing both the faulty plan and its own faulty knowledge base that allowed it to build the plan incorrectly.

The notion of learning from expectation failures is not a new one. Schank (1982) has argued that learning occurs when an understander is confronted by expectation failures. Although the failures a planner faces are planning failures, it learns from them because they are also expectation failures.

2.3 Storing Plans for Later Use
To store a plan in memory, a planner has to understand when it will be appropriate to use it again. For most planners this has meant storing plans in relation to the goals they satisfy. For a case-based planner that tries to anticipate problems and find the plans to avoid them, this is not enough. To access a plan that avoids a certain problem, the plan must have been indexed so as to allow such a connection. The basic vocabulary of plan indexing is necessarily the vocabulary of the planner's domain, and of the goals of the domain. But this vocabulary is not sufficient to allow a planner actively to avoid the problems it anticipates. Plans must also be stored by descriptions of the negative goal interactions they avoid. A plan to go outside with an umbrella has to be indexed not only by the fact that it gets the planner outside, but also by the fact that it does so while protecting him from the rain. It has to be indexed by the fact that it avoids the problem of the planner getting wet. This is so it can later be accessed when the need for such a plan
is inferred. As with any vocabulary item based on a plan satisfying a goal, the fact that a plan successfully avoids a problem must be used to index it for later use when a goal to avoid the same or a similar problem arises.

Plans are indexed by the goals they satisfy and by the problems they avoid. This allows a planner to find plans that achieve the goals it is planning for while avoiding the problems it predicts will arise when doing so.

3. LEARNING FROM PLANNING

A case-based planner must also be a learning system because it must reuse its own experiences. The learning done by a case-based planner is learning by remembering. This is a type of learning not addressed either by the theories of concept learning via induction (see Lebowitz, 1980; Michalski & Larson, 1978; Winston, 1970), or explanation-driven learning (see DeJong & Mooney, 1986; Mitchell, Keller, & Kedar-Cabelli, 1986). Case-based planning requires a knowledge-based learning that makes use of the planner's understanding of the world to determine what should be learned and when it should be learned. This learning breaks down into three types: plan learning, expectation learning, and critic learning.

Plan learning is the creation and storage of new plans as the result of planning for situations that the planner has never encountered before. The planner has to build a new plan and decide what features are best for indexing it in memory. Any new plans are stored in memory, indexed by the positive goals they satisfy as well as by the negative effects they avoid. For example, a plan for going outside in the rain to get a paper would be indexed by the fact that it is a plan to retrieve a paper and by the fact that it allows the planner to avoid getting wet.

Expectation learning is somewhat more complex than plan learning, but is closely linked to the indexing of plans in memory. It involves learning the features in a domain that are predictive of negative interactions between plan steps. This predictive ability is used to anticipate particular problems and then to search for plans in memory designed to avoid them. These features are learned by building causal explanations of planning failures and marking the states and steps that lead to the failures as predictive of them. The fact that it is raining and the planner has a goal to pick up his paper would be marked as predictive of the problem of getting wet, because these features would be included in any explanation of why the plan that lacked an umbrella failed. Once predicted, this problem could be planned for by finding a plan to avoid it. Once one of these predictions is activated, problems can be avoided by searching memory for a plan taking them into account.
Critic learning occurs when a problem in a plan can be traced back to a specific object or initial state related to a plan rather than an interaction between steps. Any repair made to a plan because of an idiosyncratic object, can be saved and associated with the object. The fix to a specific problem can become a general repair that can be applied in later cases of the problem’s occurrence. Even if an overall plan to avoid a problem cannot be found, the repair can then be reapplied to fix a plan that satisfies other goals. This would be like predicting that the rain would get someone wet in any case of going outside and then applying the repair of using an umbrella to a plan for going outside that does not already include this fix.

Case-based planning involves three types of learning:
- Learning new plans that avoid problems;
- Learning the features that predict the problems;
- Learning the repairs that have to be made if those problems arise again in different circumstances.

All three of these types of learning are supported by a planning vocabulary that describes plans in terms of the direct goals they satisfy and the interactions they deal with. Storing plans in terms of the goals they satisfy is not enough if the planner wants to reuse them. They also have to be stored in terms of the problems they avoid. This makes it possible for the planner to rediscover these past plans when it predicts the same problems again in a different planning situation.

In the following sections, I will attempt to build up a model of case-based planning that has these attributes. I start with the simplest possible model and add capabilities as needed to increase the planner’s functionality.

4. THE STRUCTURE OF CASE-BASED PLANNING

Human behavior often seems capricious. Minds wander from topic to topic. Ideas connected by thin strands of association follow one another as though intimately related. Past experiences, which have little to do with the apparent features of a new situation, present themselves as solutions to the present problem:

Any effort at cognitive modeling must begin by explaining these seemingly capricious acts. One set of approaches to this modeling task has grown out of the idea that what appears to be capricious action actually is capricious. The theories and programs that have resulted from this idea often include random number generators, random weights associative on links, or random connections between conceptual structures. They treat the seemingly random leaps of human thought as truly random, as a failure in an otherwise orderly system. Unfortunately, these theories have a seductive allure
because they present models that look very much like what they are trying to explain.

The key word here, however, is "explain." These theories look like what we want to explain, but they do not explain what we want to explain. They are mere simulations of behavior that do nothing to explain why it arises. They are "non-explanation explanations."

Another way of looking at human behavior is to ask what function this behavior serves. Instead of looking at behavior in a vacuum, examine it in terms of its possible use in performing a cognitive task. Let the function served by the behavior guide the examination of it. Make the function underlying the behavior have more importance than the simulation of the I/O of the behavior itself.

The difference between these approaches is simple. It is the difference between simulating an engine by building a box that makes engine noises and modeling an engine through an examination of the function it has and how internal combustion satisfies it.

An example of this approach is the work on naturally occurring remindings by Schank. In *Dynamic Memory*, Schank (1982) looked at the phenomena of remindings and argued for their use in learning. More recently, he has augmented this view and argued that remindings also play a role in the construction of explanations (Schank, 1984, 1986; Schank & Riesbeck, 1985). This work is aimed at a functional model of why people have these remindings and what they can do with them, Rather than a mere descriptive simulation of how to go about generating them.

The theory of case-based planning is aimed at a functional explanation of certain aspects of human cognitive behavior. In building the theory, one eye was kept on the data concerning episodic reminding and the other on the needs of a case-based planner. As a result, the theory describes how to use past plans in the construction of new ones and how to use remindings of past failures to avoid repeating mistakes.

---

The planner is reminded of past episodes for the same reason people are reminded of past episodes: These episodes contain information that can help in planning for new problems.

---

5. BUILDING IT FROM THE BOTTOM

The following sections will look at the nature of case-based planning in general, arguing why planning should make use of the case-based paradigm and then looking at what it takes to build such a planner. The simplest possible case-based planner, a plan RETRIEVER, is used to begin with, and is then expanded with the additions required to do plan modification, plan repair, and problem anticipation.
Note that I start with an intentionally simplified system and use later additions to expand its functionality. As a result, the early stages of the description lack some features that will be introduced later.

5.1 Why Case-Based?
The argument for case-based planning is straightforward: A planner that can learn and recall complex plans is preferred over one that has to repeat work it has already done. In the case of a single plan for building a car or a house, the number of steps involved is huge. Although such plans can be built up from a set of rules or from plan abstractions each time the planner needs them, it is more economical to save entire plans and recall them for reuse when situations requiring their use arise.

It is always useful for a planner to save the plans it creates, especially in those situations in which a plan includes information about how to avoid problems the planner's base of rules tended to lead the planner into. For any planning task involving the reuse of information, the best approach is to make use of a detailed representation of the experience itself. Given that all planning tasks make use of past information, this argues that the best approach to planning in general is to find and modify past plans rather than rebuild from a set of rules each time.

The functional justification for case-based planning is the need to learn from experience.

The needs of a case based planner are somewhat different from those of a rule-based system in that a case-based planner relies almost entirely on its memory of past plans. A rule-based system, on the other hand, makes almost no use of its nearly nonexistent memory. The following sections will discuss the functional needs of a case-based planner, beginning with a basic planner and then expanding it to deal with problems as they arise.

5.2 Plan Retrieval
At its simplest, a case-based planner is a memory that returns to some previously stored plan whenever it is given a new set of goals. If it cannot find a plan that satisfies all of the goals it has been handed, it returns a plan that meets most or many of them. It is trying to find what will be referred to here as the best match between the current situation and some situation in the past for which it has a plan. There is no guarantee that the best match will actually be the best possible plan for use in the current situation, but the effort is to construct a memory organization aimed in that direction. This basic planner, which only finds best matches, will be called a RETRIEVER.
Its input is a set of goals to be achieved, and its output is a plan from its memory that achieves as many of these goals as possible.

If a planner is to retrieve the right plans, the goals it is asked to achieve must be used to index past instances in memory. Planning episodes have to be indexed in memory by the goals they have satisfied. An episode in which a planner flies to London should be indexed in memory as a plan to get to London and, at a more general level, as a plan to get to someplace distant. An episode in which the planner removes an inflamed appendix should be indexed as satisfying that goal, but also by the more general result of removing a diseased organ. The simple plan of calling the hotel desk for a wake-up call should be indexed by the fact that it satisfies the goal of getting you up in the morning. A planner has to be able to discriminate among plans on the basis of all of the goals it is trying to accomplish.

Along with goals, a planner must also know what its initial planning situation is. It must know the states that are currently true, and the goals it wants to satisfy. Then plans designed for particular situations, not just particular goals, can be found and used. For example, a plan to retrieve a newspaper from the porch when it is raining, which would include the use of an umbrella, must be indexed in memory under the features that make its use relevant. This means indexing it not only by the goal it satisfies, getting the newspaper, but also by the conditions under which it is appropriate to use, namely when it is raining. Likewise, the plan to call the hotel desk for a wake-up call cannot just be indexed as a good plan for satisfying the goal to wake up at a specific time in the morning. It also has to be indexed as a good plan in the context of being in a hotel to begin with. It is not particularly effective when at home. This means having a number of plans in memory that satisfy the same goal or goals, but are distinguished by different situations. The states that define these different situations must also be used by the planner to index plans in memory.

Specific goals and states alone, however, are not sufficient to find the plans the RETRIEVER has to return. Sometimes there is no plan satisfying a particular goal, so the best match has to be a plan satisfying a similar goal. If there is no a plan to get to London, a plan that worked in getting the planner to Paris may have to be used as the planner’s starting point. Or a plan to build a two-story house may have to be modified when the planner requires a plan for a three-story building. A planner’s representation must include some notion of similarity between goals. This similarity can be expressed by placing similar goals into sets, by building them into an isa hierarchy, or by dynamically evaluating their similarity on the basis of individual features. Paris and London may both be considered as foreign countries; a gall bladder that has to be removed may be thought of as belonging to the same abstract class as an appendix; or a house with two stories may be understood in terms of structural features that are shared with a three-story building. When a
planner cannot find a plan that fully satisfies the goals it is planning for, it has to have some way to find a plan or plans that partially satisfy them. No matter what the method, there must be some metric for the similarity of goals a planner can use to judge partial matches.

But a planner has to be able to do more than just find plans. It must also be able to choose among plans. Given a set of goals, a planner needs to find the plan in memory that comes closest to satisfying all the goals. In the trivial case of having a plan, which intact, satisfies all goals, there is no problem. As soon as a planner is confronted with a set of plans that are all partial solutions, however, a problem arises: How does it determine which plan out of a group of plans best satisfies a set of goals if each of the plans satisfies some of the goals?

If goals all had the same value, the solution would be that the plan satisfying the largest number of goals would be the best match. But goals are not featureless objects all having the same value. Some goals have more value than others. The plan RETRIEVER has to know about the relative values of goals so that it can find a plan that maximizes the planner's utility rather than simply maximizing the number of goals satisfied.

One way to implement this is to store the goals themselves in a *value hierarchy* that can be used to determine the relative utilities of different plans with respect to a set of goals, (Carbonell, 1980). It is important to distinguish this from the *abstraction hierarchy* that is used to determine the similarity between plans. The abstraction hierarchy tells the RETRIEVER if a plan partially satisfies a goal; the value hierarchy tells it how much that goal is worth. The source of this hierarchy is not important. What is important is the notion of deciding among competing plans on the basis of their relative utilities, no matter how those utilities are determined.

For example, imagine a memory with two plans: one for a two-story building of glass and steel and one for a five-story building of brick. The planner is given the goal of designing a five-story building of steel and glass. All other factors being equal, a planner would have to decide which plan would be the best to modify: the plan using the materials requested or the plan having the basic structure asked for. In this case, the first plan for the two-story building can be easily modified to have extra stories; the other plan would have to be redesigned from the bottom up. The first plan would be selected on the basis of the ease with which the initial plan could be changed to accommodate all of the planner's goals. The value hierarchy of the goals would be linked to the planner's own abilities. Goals that are easier to incorporate into existing plans are less important than those that are more difficult to satisfy.

In order to get a plan that is the *best match* for a set of goals, a planner needs three kinds of knowledge:

1. A memory of plans indexed by the goals they satisfy;
2. A similarity metric for judging the similarities of goals that is required for determining how close a plan comes to satisfying a set of goals;
3. A value hierarchy of goals used to judge the relative utilities of plans with respect to a set of goals.

To integrate this knowledge (Figure 1), the RETRIEVER must function as an indexing system that uses goals and abstractions of the goals to discriminate through memory, and then use its knowledge of the relative values of the different goals to decide among the overall values of competing plans. This defines a basic case-based planner that takes a set of goals and recalls a plan from memory which satisfies as many of the most important goals as possible, thus maximizing the planner's utility.

For example, in designing a building for use as an office complex there are many goals having to do with access, available floor space and cost that have to be planned for. A plan retriever has to search for a single plan satisfying as many of the goals as possible, with the understanding that some goals are more important than others and that goals which cannot be satisfied directly, can often be partially satisfied. In this case, a past instance of a known office complex that satisfies similar goals can be found and then modified to fit the exact needs of the planner. By finding a baseline plan that satisfies some goals and partially satisfies others, the planner avoids redoing the work that was already done and stored away in memory.

To get a best match, the RETRIEVER needs a plan memory, a goal similarity metric (in the form of an abstraction hierarchy) and a goal value hierarchy.
But this model is not complete. Aside from the fact that a RETRIEVER alone is not a real planner, there is also the fact that what has been defined in this section is not quite a full RETRIEVER. It can find past plans on the basis of similarities among surface-level goals, but it cannot find past plans on the basis of interactions among those goals. A planner also needs a vocabulary to describe similarities among situations not captured by the surface features alone. This problem will be examined after some discussion of the repair mechanisms from which this vocabulary emerges.

5.3 Plan Modification

A plan retriever can only find and suggest past plans for new situations; it cannot do anything about modifying these plans to satisfy new goals. Another process, one that modifies plans to satisfy goals not already satisfied by the retrieved plan, must be added to the retrieval process. This process is called plan modification.

In order to modify a plan, a planner needs a variety of information. It needs a library of modification rules that are designed for plans and classes of goals. These rules will be sets of steps that can be added to particular plans to achieve particular goals. These modification rules do not have to be complete plans for achieving any particular goal, they can just be the modifications needed to alter an existing plan in order to achieve that goal. Access to these modification rules will allow a MODIFIER to add new steps to a plan in a way that is sensitive to the type of plan being modified and the particular goal that is being satisfied. The MODIFIER also needs information about specific objects in its domain that outlines how it should accommodate specific features of these objects while using its more general rules. This information, in the form of special purpose critics, will let it tailor the general modifications of a plan to the specific needs of the items required in order to achieve particular goals. Finally, the MODIFIER needs to know about what the plans it is modifying are supposed to be doing in general. This is needed so that it does not violate the goals of the overall plan when it modifies it to satisfy a specific goal.

To alter old plans to meet new goals, the MODIFIER needs a set of modification rules, critics with knowledge of goal-specific requirements and general plan specifications.

Thinking back to architecture, this means that to add a window to an existing design, the MODIFIER would have to know the goal to be achieved (an added window) and the type of design being altered (an office, apartment or house). It would have to take into account the features of the particular window (the type of glass, the size, or shape). The changes required for adding a window to the design for an apartment building are different.
from those made to a plan for a standard house. By storing modifications in terms of both the goal to be added and the type of plan being altered, a MODIFIER can be sensitive to these differences. By also storing idiosyncratic steps dealing with the features of particular items in a domain, it can deal with those items within the context of a more general process of plan alteration.

Another example is in the domain of automotive design. Imagine a situation in which one of a planner's goals is to have different parts of a car it is designing be colored red. If a part is an exterior metal piece, the alteration to the initial plan will involve changing the color of the paint used to cover the part. If the part is an interior plastic part, the change will involve altering the pigments used in the initial mixing of the plastic. The different initial plans determine different alterations in response to the same goal. No one plan for changing the color will do. Different alterations, associated with the different initial plans, have to be used for the different situations.

The RETRIEVER and MODIFIER are the core of a case-based planner. The RETRIEVER takes a set of goals and finds the past plan that best satisfies them. The MODIFIER takes the plan and the goals it fails to meet and modifies the plan to satisfy all of the goals it has been given. To do this it needs a set of modification rules, critics with knowledge of goal-specific requirements and general plan specifications (Figure 2). For a given set of goals, the RETRIEVER finds a good plan and the MODIFIER makes it better.

5.4 Plan Storage
The idea of base-cased planning was initially justified by the need to learn. The whole notion of this kind of planning rests on the desire to alter the
planner's abilities on the basis of its own experience. Within the confines of the planner built so far, only one kind of learning is really possible. This is learning by remembering; that is, learning a new plan that has been built by storing it, along with existing plans, in the planner's memory.

The features used to store a plan must be the same as those used to access it. The knowledge a planner needs to store a plan is exactly parallel to the knowledge it needs to find one. In part, the indices used to store plans are the goals that the plan it is storing satisfies. A planner can identify these goals using the general goal information associated with the type of plan it is building and the specific goals satisfied by the particular plan it is storing. General plans have descriptions of the nature of the goals they satisfy.

For example, the general notion of a plan for an electrical circuit includes the idea that the input and output behavior of the circuit is important. It also includes the notion that the cost, size, and durability of actual implementations are important. A particular circuit will have particular characteristics as well, such as its color or its overall aesthetic character. However, only those features that relate to the goals of circuits in general or to the planner's own immediate goals are used to store the particular plan for later use. The features not relating to the goals satisfied by the plan will not be used to index the plan for a particular circuit in memory. Later, when a new set of goals is being planned, the planner can use its current goals to search for a plan in memory to satisfy them.

By using only the goals the plan was designed to meet, rather than the entire set of states that are true as a result of running the plan, a planner can limit the features it uses to index a plan in memory while still guaranteeing that the plan is indexed under all the features that are used to retrieve plans.

But a planner cannot index plans only by the goals they satisfy. As in the RETRIEVER, it has to attend to the circumstances under which they do so. A plan to use an umbrella to stay dry is a great plan to use when it is raining, but a less than effective one to use when the planner wants to go into a flooded basement. Plans and planning episodes have to be distinguished by the goals they satisfy and by the circumstances under which they do so. The circumstances effecting the choice of a plan in the initial construction have to be included in the indexing of the plan in memory.

The STORER aids in the building of later plans by storing the work that has been done by the RETRIEVER and the MODIFIER (Figure 3). The STORER does nothing to help in the building of a present plan. The job of the STORER is not to alter the plan that has been built. Rather, it must alter the memories of the planner itself, giving it access to a complete plan, which previously had to be built from another plan, and from the application of the planner's modification rules. Although this only means a savings of time at this stage, later, when the planner is given the ability to recover from its own failures, it will mean that the planner will be able to use memories of past plans to avoid repeating mistakes that waste other resources.
The vocabulary used by the RETRIEVER and STORER to access plans in memory is not yet complete. In particular, the vocabulary currently lacks any way to describe plans in terms of the problems they avoid. Both the STORER and the RETRIEVER require access to this vocabulary, giving the STORER the ability to place a plan in memory indexed by the fact that it avoids certain problems, and giving the RETRIEVER the ability to find plans when it anticipates the need to avoid the same problems. I will hold off discussion of this vocabulary, however, until after outlining the process that gives rise to it: plan repair.

The learning by remembering that the STORER does it not the only kind of learning that will be done by the planner being built. This kind of learning will be augmented with a module that does two other types of learning: It will learn the features in a situation predicting problems, and it will learn new rules for adapting particular items to the general modification plans associated with the plan types. This planner will not only remember new plans, it will also learn to anticipate problems and build rules for adding new goals to already existing plans.

To place new plans in memory, the STORER needs to index them under the same features the RETRIEVER uses to find them: the goals they satisfy and the situations in which they are appropriate.

A basic case-based planner must incorporate these three processes:

1. A plan retriever that finds a plan from the past that is a "best fit" with the current situation;
2. A plan modifier that can alter a plan with changes that are responsive to the goal being added and the plan being altered;
3. A plan storer that can place a complete plan into memory indexed by the goals it satisfies and the conditions under which it does so.

5.5 Plan Repair
For a planner to be practical, it has to be able to repair failed plans. No matter how good the planner is, at one time or another it will have to confront problems that arise out of its own lack of knowledge and the limits of its own heuristics. Given that the planner is going to make mistakes, it has to have some mechanism for repairing the faulty plans it builds. This mechanism will be called the REPAIRER.

The input to the REPAIRER has to have two parts: a faulty plan and some description of the fault itself. In other words, the input must include the plan and either the desired state it has failed to achieve, or the undesired state it has caused to come about. How a planner gets this information can vary. It can actually run its plans and examine the results. It can run simulations of the plans and use their results to diagnose errors. It can even ask an outside source if the plan will do what it wants it to. But no matter how it does it, a planner has to be able to notice and respond to its own failures.

The basic approach taken here is to diagnose first and then repair. The REPAIRER is going to have to have some vocabulary for describing plan failures that can be used to index methods for repairing the plan itself. This vocabulary can be a simple statement of the fact that the plan is faulty. Unfortunately, a statement like “The plan has failed,” does not provide much guidance as to how to go about repairing it.

Given that a planner has to understand the goals it is trying to achieve and the states associated with these goals in order even to recognize planning failures, it should, therefore, also have access to its vocabulary of goals in order to describe those failures. This would allow the planner to make statements to itself, like “The engine failed to start when I turned the key.” This vocabulary, however, does not capture all of the information that can be used to access appropriate repair methods, because there could be many ways in which the states constituting the failure could have come into being. A single type of failure, such as an engine failure, can be the result of many different situations, so the information about the failure of the goal provides only a little guidance as to how to deal with the problem. The best it can give is a method for responding to the failed goal in general, which may or may not be appropriate to the facts of the matter. For example, an engine may fail to start because there is no gas in the car, or the battery is dead. A vocabulary that includes only the fact that the engine has failed to start could not be used to find the best strategy for dealing with the problem.

A more extensive vocabulary must include constructs for describing why a failure has occurred, along with those for describing the failure itself. This would mean forming descriptions like “The engine failed to start because a
wire leading to the starter has shorted out on the body where it passes behind the side-mounted air filter; the heat exchange from the air filter melted the insulation on the wire." The explanation, because it points to the states and actions participating in causing the failure, provides the focus as to what parts of a design have to be changed. Even if a specific method for fixing the particular problem does not already exist, one can be generated out of a method for dealing with the general problem (a side effect of one part violating the maintenance conditions of another) and the particular states (the side effect is excess heat, the maintenance condition is the fact of the insulation and so on). This could be used to suggest general methods that could be instantiated, using the specific facts of the current problem. This would allow a response like recovering from the side effect (find a way to drain off the heat from the air filter) or altering the part being interfered with, in order to compensate for the presence of the side effect (use a heat-resistant insulation on the wire) or changing the initial plan being interfered with (reroute the wire altogether).

No matter what vocabulary the REPAIRER uses, the idea is the same: Describe the failure and use this description to find methods for dealing with it. Along with the vocabulary for describing planning failures, the REPAIRER needs a set of repair methods that can be accessed with that vocabulary. These methods should be organized so that the description of a given failure will access those, and only those repair methods, that have a chance of repairing that particular plan fault. This organization is the main reason that the REPAIRER describes the problem at all. This relationship between problems and repairs is like the relationship between goals and plans. Plans are indexed under the goals they satisfy and repair methods are indexed under the types of problems they resolve.

A plan repairer needs access to two types of knowledge: a vocabulary describing plan failures, and a set of repair methods corresponding to those descriptions. With these, it can describe problems and then use those descriptions to access the strategies for dealing with them.

There is a learning aspect of the REPAIRER that goes beyond the confines of the single plan it is repairing. That is, once the plan is repaired, it is not only a plan that satisfies a set of goals, it is also a plan that avoids a particular problem. The goals the planner was originally planning for have interacted to cause a failure. The repaired plan, because it is the repaired version of the failed plan, is now designed to cope with that interaction between goals and to avoid the failure altogether. If the REPAIRER allows the STORER to store the new plan only by the goals it satisfies directly, it will lose the information that the plan is also one to use if the planner is trying to avoid a particular problem.

The failures a plan avoids while satisfying other goals cannot be identified by looking at a finished plan without examining the reasoning that went
into constructing it. It is only during the planning process, when the plan fails and is repaired, that the problems arising out of certain interactions can be identified. The fact that a plan avoids a particular problem can only be seen by the REPAIRER, and the REPAIRER then has to tell the STORER about the goals the repaired plan satisfies and the failures it avoids along the way. Given this added vocabulary, the STORER can place successful plans in memory, under the goals they achieve and the problems they avoid.

With the addition of the REPAIRER, the STORER can now index plans by the problems they avoid as well as the goals they satisfy.

The REPAIRER is invoked only when a plan fails. Its task is to repair the plan and tell the STORER how to characterize it so that it can be found again in a similar problem situation. The REPAIRER is called only after a plan has been modified and run, only after the plan has been committed to by the RETRIEVER and MODIFIER. It requires a vocabulary for describing planning problems and a set of strategies that are indexed by the descriptions of the problems they solve. Once it repairs a plan, it then has to hand it to the STORER for placement in memory, indexed by the goals the plan satisfies and the problems it avoids (Figure 4).

To repair failed plans and describe them to the STORER, the REPAIRER requires a vocabulary of plan failures and repair strategies indexed by that vocabulary.

5.6 Learning from Failure
A case-based planner indexes the plans it builds by the problems it avoids while achieving its goals, but there is still a stumbling block to reusing that plan for similar problems in the future. This is the planner's ability to figure out when the problem is going to arise again.

The fact that a plan solves a particular problem is not useful unless the planner can anticipate that problem in the appropriate circumstances and use that prediction to find the plan in memory. The usefulness of a plan that solves a problem rests not only on having the plan indexed by the fact that it does so, but also on the ability to predict that the planner is going to need a plan to solve that particular problem. Having a plan to solve a problem does the planner no good if it cannot recognize the circumstances in which that problem will arise. The ability to predict when a problem is going to arise in the future, however, rests on the ability to figure out why it happened in the past. To do that, the planner needs a function that can decide which features of a failed plan caused the failure to occur. Then it can extrapolate from these to the features in later situations that will predict when the problem
will arise again. This function, which does the credit assignment as to which features are to blame for a failure, is called the ASSIGNER.

The job of the ASSIGNER is to look at a failed plan and decide what circumstances will be predictive of that failure in the future. The knowledge it uses can vary in much the same way that the knowledge used by the REPAIRER can vary: It can be simple and unreliable, or complex and robust.

To decide which features in a situation are to blame for a failure, the ASSIGNER needs to be able to describe the causes of the failure. The more extensive its vocabulary for this description, the more exact its credit assignment will be.

Take, for example, a situation in which a plane traveler has missed a ride to his hotel with a fellow traveler because he has had to wait for his baggage to come off the plane. Here is a case where a failure has occurred that can be related to the features predicting it: checking baggage while traveling with others. When later performing a similar plan, that is, taking a flight
with other people and checking baggage, a planner that can recall the failure and now plan for it by taking only carry-on luggage will be in a better position than one that could not anticipate the problem.

In the example of the failed engine, the fact that a new design for a side-mounted air filter caused a set of wires to be routed through an area of high heat, should be assigned the blame for the problem. If this problem arose out of the combination of default plans, this assignment has to be such that when the goals associated with those plans are asked for again (such as the goal for a side-mounted air filter), the problem can be predicted and thus avoided.

No matter what the method, the ASSIGNER’s task is to mark features in a situation as predictive of the problems that arose in that situation. Like the STORER, its function has no effect on problems the planner is currently working on. Instead, its job is to assure that the planner is able to predict when the current problem is going to arise again in later circumstances. Its output can be a set of inference rules fired in the early stages of planning, links going from surface goals to predictions, or a table of effects matching features to predictions. The form of the output is not the issue here. The issue is that the planner, because the ASSIGNER is able to identify the features predicting a problem, is now able to anticipate that problem and use the goal of avoiding it to find a plan that does so. While it looks at problems along with the REPAIRER, its output is not a plan, but is instead a knowledge base of possible problems that can arise, and the circumstances that predict them (Figure 5).

5.7 Problem Anticipation

When a plan fails, the features that participated in that failure are built into a base of rules or connections, which allows the planner to anticipate similar problems in the future. Another module, which anticipates problems on the basis of these features, has the task of taking these rules or connections and making the predictions.

The job of an ANTICIPATOR is to look at the planner’s goals and the situation surrounding them to decide if there is anything in the situation predictive of a problem, before any other planning is done. This is so the prediction of a problem can be used to find the plans in memory that avoid it. The whole point of an ANTICIPATOR is to provide information about problems that have to be avoided, information that will be used by a RETRIEVER to find a plan to do so; it makes sense that an ANTICIPATOR has to be called prior to any search for plans (Figure 6, p. 408). This problem anticipation is done prior to the initial search for a plan in order to give the planner the information about what problems it has to avoid so it can search for a plan that does so.

The knowledge it uses is the base of information about the features which predict problems the ASSIGNER has built up as a result of examining the
Figure 1. The ASSIGNER.

planner's failures. The more failures a planner has, the better it becomes at anticipating problems and thus avoiding them.

To anticipate a problem on the basis of surface features, the ANTI-CIPATOR needs the base of information built by the ASSIGNER.

Consider again the example of the traveler. The task of the ANTICIPATOR is to take the features of the situation the planner is dealing with, and recall the problems that have arisen in past situations to handle them. By being reminded of the problems caused by waiting for luggage, a planner can either find a past plan dealing with the problem, or alter another plan in response to the prediction. The problem can be avoided because it can be predicted.

Likewise, when the goal to have a side-mounted air filter arises again, the prediction of the problems resulting from the modifications required to satisfy this goal can be used either to find the past plan that already dealt with the problem, or the past repairs that were made, which could also be
used in this plan so that the problem is avoided. In this way, problems that have been encountered before can be avoided. This is in contrast to a procedure in which plans are built without attending to past problems, a procedure in which the problems are repeated and must be repeatedly repaired.

There is an important relationship between plans stored in memory and tasks of the ANTICIPATOR and the RETRIEVER. Once a problem plan has been repaired, it is stored in memory, indexed by the fact that it deals with a particular problem. At this point, however, there is no way for the planner to look at a new situation and predict that it will have to avoid that problem. So there is no way for it to find the plan in situations where the problem will come up again. Because of this, it will continue to make the same mistake because it does not know that the planning approach it used before in a similar situation led to problems. If it can figure out the causes of a failure, however, it can use this information to anticipate the problem in similar situations and look for a plan that avoids it. The ASSIGNER figures out the causes of problems. The ANTICIPATOR then uses the
information built up by the ASSIGNED to predict the problem again when similar causes are present. The ANTICIPATOR notices that a problem is going to arise and then tells the RETRIEVER to find a plan to avoid it.

This communication from ANTICIPATOR to RETRIEVER is important in that it gives the RETRIEVER the added vocabulary for describing planning situations the STORER is using to place plans in memory. The STORER, because the REPAIRER can tell it what problems a plan solves, is able to place plans in memory, indexed by the facts that they do so. The RETRIEVER, because it has the ANTICIPATOR to look at the surface features of a situation and provide predictions concerning the same problems, is now able to request plans avoiding those problems. So plans that deal with particular planning difficulties due to the interactions between plan steps can be stored and retrieved, indexed by the fact that they do so. The STORER uses information about the problems the plan solves to put it into memory and the RETRIEVER uses the predictions of reoccurrences of those problems to get them back out.

With the addition of the ANTICIPATOR, the RETRIEVER can search for plans on the basis of the problems they avoid, as well as the goals they satisfy.

A final point about the ANTICIPATOR: Anticipation plays a role in case-based planning similar to that of projection in more traditional systems. They both serve to warn the planner of difficulties that might arise in running the plan. There is a substantial difference between the two concepts, however. Projection makes use of all of the planner's domain knowledge to simulate and thus predict any problems that might come up in a plan. Anticipation makes use of the planner's experience with failures to make this prediction. The difference is not a subtle one: Projection involves exploring all possible interactions among the results of plan steps, while anticipation involves exploring only those possibilities that have actually given rise to problems in the past.

Given Chapman's (1985) recent result concerning the complexity of projection, it seems clear that projection in its current form is an untenable approach to teasing out planning problems. This need for a new view of prediction, one that is more heuristic in nature, only serves to confirm the initial belief that anticipation on the basis of known failures is a viable alternative to the more complete, yet far less tractable, approach of projection.

5.8 The Final Package
The basic case-based planner, which grows out of the need to reuse plans and adapt them for new goals, functions as follows: A set of goals is handed to the planner and sent directly to the ANTICIPATOR. The ANTICIPATOR, based on the knowledge built up by the ASSIGNED, makes any predictions
of planning problems it thinks will arise out of the current goals. The goals are then handed to the RETRIEVER along with the ANTICIPATOR's predictions of problems to be avoided. The RETRIEVER uses both to search for a plan in memory that best satisfies the goals it is trying to achieve and avoid any anticipated problems. The result is a past plan that matches some or all of the goals now being planned.

This plan is sent to the MODIFIER, which adds or substitutes new steps to the plan in order to make it satisfy any of the planner's goals it does not yet achieve. Once modified, the plan is run and the results checked against the goals of the planner. If there is a failure, either because a desired goal has not been achieved, or because an undesired state has resulted from the plan, the plan is given to the REPAIRER.

The REPAIRER builds a characterization of the failure and uses this to find and apply one of its repair methods. The repaired plan, along with a description of the problems that had to be solved along the way, are then sent to the STORER for placement into memory. The STORER indexes the new plan by the goals it achieves and the problems it avoids. Now the plan can be used again in similar circumstances in the future.

While the REPAIRER is repairing the plan, the ASSIGNER is deciding which features in the original request, that is, which goals, interacted to cause the failure to occur. Once it has done this, it marks these features as predictive of the problem so that the ANTICIPATOR can anticipate the problem if it encounters the goals in a later input.

This planner does two things as it builds a plan: It tries to satisfy a set of goals using its model of what plans are appropriate for different situations, and at the same time, it is also tests that model for the appropriateness of those plans against the real world, so that later planning will be easier and more reliable.

To serve its first function, the planner has to react to failures by repairing the present plan. To serve its second function, it has to alter its view of the world by adding new plans indexed by the problems they solve and by altering its predictions so that it can anticipate those problems and find the plans to avoid them.

The fact that planning and learning are so intimately connected in case-based planing is no accident. The power of a case-based planner is directly dependent on its ability to reuse plans, and the only way to reuse plans effectively is to take seriously the notion of learning, which features in a planning situation determine when they are appropriate to use.

6. LEARNING FROM PLANNING

A case-based system learns by planning. It learns by storing the results of its own planning experiences in memory. This makes it a bit odd to talk about some aspects of learning from planning. A case-based planner does learn
new plans, but learning a new plan actually involves deciding which features are important for indexing a plan it has already built. The learning is not in the building, it is in the storing and indexing of things that already have been built.

A case-based planner learns by correctly indexing its planning experiences in memory.

A planner can improve by recalling its past experiences and anticipating problems it has encountered before, so that they can be avoided. The task of learning in this case is not building the failure, it is deciding which features caused it in the original case, and thus, which features will predict it in later cases. By linking the features which are predictive of a failure to the memory of it, the planner can anticipate problems before they occur, and use this anticipation to retrieve a plan in memory to solve them.

If a surgeon encounters a problem while performing an operation, such as a patient’s low blood pressure being reduced even further by a particular anesthetic, thus leading to heart failure, it makes sense for him to learn something from this failure. In particular, it makes sense for him to learn that patients with low blood pressure will go into cardiac arrest if this anesthetic is used on them. By linking these features to the memory of the failure, the surgeon will be able to anticipate the problem when it arises and use this prediction to find a plan to avoid the problem.

Memories of past planning experience can also be helpful to a planner in dealing with the situation in which it predicts a failure but is unable to find a plan that deals with it. In these situations, the memory of the change that was made to repair a past plan in response to the same failure can often be used to repair the current plan before it is run. Just as the anticipation of a failure can point the planner to a past plan dealing with it, in the absence of that plan it could point the planner to a past repair that could be applied to whatever new plan is being devised. The problem of recalling past repairs is somewhat different than recalling past plans, in that not all repairs can be transferred between plans. Like memories of plans, memories of past repairs have to be indexed by the failures they deal with, but they also have to have some provision for the fact that not all repairs to a problem can be transferred for use on all plans in which that failure arises.

Even in this case, the problem is not to construct a new piece of information. The repair itself is constructed by the planner. It is only then that any learning mechanism comes into play. Here again, the job of the learner is to choose the features that will be used to index the information. A repair, such as using an alternate anesthetic when performing appendectomies, can be stored as a general repair to other plans that will be performed on low blood pressure patients. The overall appendectomy plan is saved so that it can be used again, but the specific repair of altering the anesthetic is saved.
as well, so that it can also be used to repair other plans in which the same problem is predicted.

In all three of these learning situations, the main task of the learner is to figure out which features a piece of information should be indexed under. Any generalization that is done takes the form of generalizing these features without changing the specificity of the information being stored. The plans placed in memory are the actual plans created by the planner. They are not generalized versions of these plans. The goals used to index them, however, are generalized so that the plans can be found in similar situations, if not identical, to those in which they were originally constructed.

The theory of case-based planning presented here is a theory of learning in each of these three areas. It is a theory of learning new plans, new problems, and new solutions. It is a theory of learning within the context of active planning.

A case-based planner learns new plans, the features predicting failures, and past repairs to faulty plans it can reuse. This learning is accomplished by saving the different results of the planner's own experience.

7. CHEF: A CASE-BASED PLANNER

The theory of case-based planning in this article is implemented in the computer program CHEF. CHEF is a case-based planner. CHEF's approach to planning is to make use of memory whenever possible. It begins planning by using its memory of past failures to warn of problems. It then uses its memory of successes to find the best plan to modify for its current goals. If it has a plan failure, CHEF treats it as a failure of its understanding of the world, and explains the failure so that it can repair the plan and correct the knowledge of the world that allowed it to create the plan. Finally, it saves its successes in memory, indexed by the goals they satisfy and the problems they avoid, so that they can be used again to satisfy similar goals and avoid similar problems.

CHEF's domain is Szechwan cooking. Its task is to build new recipes on the basis of a user's requests for dishes with particular ingredients and tastes. Because it is building recipes, it has a constraint many planners do not: It has to build integrated plans for multiple goals in which all of the goals are satisfied by a single plan. This is similar to the constraint in many design domains, such as industrial design or circuit construction, in which a single object has to meet a set of specifications.

Here is an example of CHEF planning for the problem of building a stir-fried dish with beef and broccoli. In the pages that follow, all text ruled on top and bottom is actual output produced by the CHEF program.
The input to CHEF is a set of goals to be satisfied with a single integrated plan. In this case CHEF is given three goals: the goal to have a stir-fried dish, the goal to include beef, and the goal to include broccoli. Instead of searching for individual plans to satisfy each of the goals separately and then merging them, as most hierarchical planners would do, CHEF searches its memory for a single plan that matches, or partially matches, as many of the goals as possible. In cases where multiple plans match its goals, CHEF chooses one of the plans at random.

Searching for plan that satisfies —
- Include beef in the dish.
- Include broccoli in the dish.
- Make a stir-fry dish.

Found recipe — REC2 BEEF-WITH-GREEN-BEANS

Recipe exactly satisfies goals —
- Make a stir-fry dish.
- Include beef in the dish.

Recipe partially matches —
- Include broccoli in the dish.
in that the recipe satisfies:
  - Include vegetables in the dish.

CHEF begins planning by finding a single plan that satisfies as many of its active goals as possible.

Once the baseline plan is found, it is modified to match whatever goals it does not already satisfy. The modification is controlled by rules related to the kind of dish being designed (in this case a stir-fry dish), and by the goal that the original plan is being modified to satisfy (the goal to include broccoli). This kind of knowledge in cooking is similar to the knowledge in other design domains such as architecture. In architecture, instead of knowing how to modify stir-fry plans, an expert knows how to alter plans for houses and apartment buildings. Likewise, instead of knowing how to add broccoli or beef to an existing recipe, an expert would know how to add windows or doors to an existing plan.

In this case, the fact that there is a partial match between the target goal, of including broccoli, and an object in the existing recipe, green beans, tells the planner that it can replace the green beans with the broccoli directly. Other structures, called object critics, allow it to correctly reduce the cooking time of the vegetables to account for the difference between the requirements of green beans and broccoli. They also tell the planner that it has to chop the broccoli before stir-frying it.
Building new name for copy of BEEF-WITH-GREEN-BEANS
Calling recipe BEEF-AND-BROCCOLI

Modifying recipe: BEEF-AND-BROCCOLI
to satisfy: Include broccoli in the dish.

Placing some broccoli in recipe BEEF-AND-BROCCOLI

— Considering ingredient-critic:
Before doing step: Stir fry the —Variable—
do: Chop the broccoli into pieces the size of chunks.
— ingredient-critic applied.

CHEF alters old plans to satisfy new goals using a set of modification rules and a set of object critics.

Once these modifications are made, the planner has a plan that should satisfy all of the initial goals it was asked to plan for. This plan has the same form as the initial case retrieved from memory: a set of ingredients (or props), a set of actions to be taken, and a set of goals the plan should now satisfy.

BEEF-AND-BROCCOLI

A half pound of beef
Two tablespoons of soy sauce
One teaspoon of rice wine
A half tablespoon of corn starch
One teaspoon of sugar
A half pound of broccoli
One teaspoon of salt
One chunk of garlic

Chop the garlic into pieces the size of matchheads.
Shred the beef
Marinate the beef in the garlic, sugar, corn starch, rice wine and soy sauce
Chop the broccoli into pieces the size of chunks.
Stir fry the spices, rice wine and beef for one minute.
Add the broccoli to the spices, rice wine and beef.
Stir fry the spices, rice wine, broccoli and beef for three minutes.
Add the salt to the spices, rice wine, broccoli and beef.

There is more to CHEF’s knowledge of this plan than just the steps that are included in it, however. It also knows about the goals that should be satisfied when it is run. These goals are derived from its knowledge of stir-
frying and its understanding of what items are important in this recipe. So it has the understanding that running this plan should result in the following:

<table>
<thead>
<tr>
<th>The beef is now tender.</th>
<th>The dish now tastes salty.</th>
</tr>
</thead>
<tbody>
<tr>
<td>The dish now tastes savory.</td>
<td>The dish now tastes sweet.</td>
</tr>
<tr>
<td>The broccoli is now crisp.</td>
<td>The dish now tastes like garlic.</td>
</tr>
</tbody>
</table>

CHEF derives the goals that a plan should satisfy from general plan specifications and the particulars of the current plan.

But not all plans work the first time. In this plan, the goal to have the broccoli be crisp will fail, because it is being stir-fried with the beef, and will thus be cooked in the liquid that the beef produces. This was not problematic in the original BEEF-AND-GREEN-BEAN recipe because green beans are a more robust vegetable than broccoli.

### 7.1 When Plans Fail
When a plan fails it is because the planner's understanding of the world has failed. It has expected that the plan would have a result that it has not had. So the explanation of a failure serves two functions: It identifies the parts of the plan that have to be altered, and it identifies the places where the planner's knowledge is faulty.

CHEF discovers a failure by monitoring a simulated execution of the plan it has produced. This simulation is the program's equivalent of real-world execution. The simulator consists of a set of inference rules used to determine the outcome of each step of the plan. Once the plan has been run, CHEF checks the final states against its goals, looking for any failures to achieve its positive goals, as well as any instances of negative states it would like to avoid.

Checking goals of recipe—BEEF-AND-BROCCOLI

Recipe—BEEF-AND-BROCCOLI has failed goals.

The goal: The broccoli is now crisp.
is not satisfied.
It is instead the case that: The broccoli is now soggy.

Unfortunately: The broccoli is now a bad texture.
In that: The broccoli is now soggy.

Changing name of recipe BEEF-AND-BROCCOLI to BAD-BEEF-AND-BROCCOLI
CHEF checks the goals of a plan against the results of a simulation of the plan that is its equivalent of the real world.

A plan failure such as this means two things to CHEF. First, it must repair the plan itself, changing the steps so that the failure does not recur. Second, it has to repair its understanding of the world, an understanding that has been shown to be faulty in light of this failure. For both of these tasks it is important that CHEF understands exactly why the failure has come about. It is not enough that it can point to the fact that the broccoli is not crisp and call this a failure. It must also be able to explain why this particular failure has happened. It has to understand the series of steps and states that led to the failure in order to alter its plan and its understanding of the world. This explanation gives it a description of the problem that can then be used to access repair strategies fitted to that problem. Because a causal explanation is used to describe the problem, a set of fixes aimed at altering the causality behind the failure can be used instead of weaker methods, such as backup or total replanning.

7.2 Explaining Plan Failures
The explanation of a failure provides a causal description of why it has occurred. An explanation describes the state that defines the failure, the step that resulted in it, and the conditions that had to be true for it to come about. It is also a description of what the planner was trying to do when the failure occurred. It includes a description of the goals being served by the steps and states that caused the failure and even those being served by other steps in the plan that may have participated in the failure. It identifies what has to be changed in the plan in order to solve the present problem, and also identifies what has to be changed in the planner's knowledge of the world so that the failure will not occur again.

CHEF explains failures using a set of inference rules to tell it about the effects of each step in its domain on each object in its domain. It uses these rules to determine what the nature of the failure is, why it has occurred and what goals are served by the steps and states that led to the failure. These rules are the same as those used to simulate the plan, meaning that CHEF can always explain the failures it encounters. These rules are not used in the initial stage of anticipation because of the computational complexity of nonheuristic projection.

The inference rules are used to chain through the steps and states of the plan to answer a set of questions that will form the explanation of why the failure has occurred. Each of these questions is associated with a particular point in the causal chain explaining the failure. Each answer is a step from the original plan, or a state caused by one of these steps. These answers are used to fill in a causal framework that will later be used to find a set of repair strategies for fixing the plan.
Explaining the following failures:

It is not the case that: The broccoli is now crisp.
in that: The broccoli is now soggy.
The broccoli is now a bad texture.
in that: The broccoli is now soggy.
In: BAD-BEEF-AND-BROCCOLI

ASKING THE QUESTION: 'What is the failure?'
ANSWER—The failure is: It is not the case that: The broccoli is now crisp.

ASKING THE QUESTION: 'What is the preferred state?'
ANSWER—The preferred state is: The broccoli is now crisp.

ASKING THE QUESTION:
'What was the plan to achieve the preferred state?'
ANSWER—The plan was: Stir fry the sugar, soy sauce, rice wine, garlic, corn starch, broccoli and beef for three minutes.

ASKING THE QUESTION:
'What were the conditions that led to the failure?'
ANSWER—The condition was: There is thin liquid in the pan from the beef equaling 4.8 teaspoons.

ASKING THE QUESTION
'What caused the conditions that led to the failure?'
ANSWER—There is thin liquid in the pan from the beef equaling 4.8 teaspoons was caused by: Stir fry the sugar, soy sauce, rice wine, garlic, corn starch, broccoli and beef for three minutes.

ASKING THE QUESTION:
'Do the conditions that caused the failure satisfy any goals?'
ANSWER—The condition: There is thin liquid in the pan from the beef equaling 4.8 teaspoons is a side effect only and meets no goals.

ASKING THE QUESTION:
'What goals does the step which caused the condition enabling the failure satisfy?'
ANSWER—The step: Stir fry the sugar, soy sauce, rice wine, garlic, corn starch, broccoli and beef for three minutes.
enables the satisfaction of the following goals:
The dish now tastes savory.
The beef is now tender.

At this point, then, CHEF knows what steps and states combined to cause the current failure. Using its inference rules for the domain, it has
been able to construct a causal chain to explain why the failure occurred. It also knows which goals were being pursued in taking the actions and creating the states that led to its failure.

**CHEF** explains the failures that it encounters through a causal description of why they have occurred.

### 7.3 Plan Repair Strategies and TOPs

This explanation serves two functions. For the short-term task of repairing the plan, it describes the planning problem in a general causal vocabulary that can be used to access some equally general plan-debugging strategies. Repair techniques are indexed in memory under causal descriptions of the situations they are built to handle. For the long-term task of altering the planner's approach to later problems so it will not repeat its current mistake, the explanation serves to point out which features in the domain interact to cause this sort of failure to occur.

The explanation of the failure is used to find a structure in memory that organizes a set of strategies for solving the problem described by the explanation. These structures, called Thematic Organization Packets, or TOPs (Schank, 1982), are similar in function to the critics found in HACKER (Sussman, 1975) and NOAH (Sacerdoti, 1975) and the goal/plan problem descriptions found in PAM (Wilensky, 1978). Each TOP is indexed by the description of a particular type of planning problem and each organizes a set of strategies for dealing with that type of problem. These strategies take the form of general repair rules such as **REORDER STEPS** and **RECOVER FROM SIDE EFFECTS**. Each general strategy is filled in with the specifics of the particular problem to build a description of a change in the plan that would solve the current problem. This description is used as an index into a library of plan modifiers in the cooking domain. The modifications found are then tested against one another using rules concerning the efficacy of the different changes, and the one that is most likely to succeed is chosen.

The idea behind these structures is simple. There is a great deal of planning information related to the *interactions* among plans and goals. This information cannot be tied to any individual goal or plan, but is instead tied to problems arising from their combination. In planning, one important aspect of this information concerns how to deal with problems due to the interactions between plan steps. Planning TOPs provide a means to store this information. Each TOP corresponds to a planning problem due to the causal interaction between the steps and the states of a plan. When a problem arises, a causal analysis of it provides the information needed to identify the TOP that actually describes the problem in abstract terms. But the TOPs are not there just to describe problems. Under each TOP is a set of strategies designed to deal with the problem the TOP describes. By finding the
TOP that relates to a problem, a planner actually finds the strategies that will help to fix that problem.

In this example, the explanation points to **SIDE-EFFECT:DISABLED-CONDITION:CONCURRENT**, a TOP related to the interaction between concurrent plans in which a side effect of one violates a precondition of the other. This is because the side effect of liquid coming from the stir-frying of the beef is disabling a precondition attached to the broccoli stir-fry plan that the pan being used is dry.

The causal description of the failure is used to access this TOP out of the 20 the program knows about. All of these TOPs are associated with causal configurations leading to failures and store strategies for fixing the situations they describe. For example, one TOP is **DESIRED-EFFECT:DISABLED-CONDITION:SERIAL**, a TOP that describes a situation in which the desired effect of a step interferes with the satisfaction conditions of a later step. The program was able to recognize that the current situation was a case of **SIDE-EFFECT:DISABLED-CONDITION:CONCURRENT** because it had determined that no goal is satisfied by the interfering condition (the liquid in the pan), that the condition disables a satisfaction requirement of a step (that the pan be dry) and that the two steps are one and the same (the stir-fry step). Had the liquid in the pan satisfied a goal, the situation would have been recognized as a case of **DESIRED-EFFECT:DISABLED-CONDITION:CONCURRENT**, because the violating condition would actually be a goal-satisfying state.

```
Found TOP TOP1 — SIDE-EFFECT:DISABLED-CONDITION:CONCURRENT
TOP — SIDE-EFFECT:DISABLED-CONDITION:CONCURRENT has 3
strategies associated with it:
  SPLIT-AND-REFORM
  ALTER-PLAN:SIDE-EFFECT
  ADJUNCT-PLAN
```

CHEF uses its causal description of a problem to find a TOP with strategies that can solve it.

The TOP, **SIDE-EFFECT:DISABLED-CONDITION:CONCURRENT**, has three strategies associated with it: **SPLIT-AND-REFORM**, which suggests breaking the parallel plans into serial steps: **ALTER-PLAN:SIDE-EFFECT**, which suggests using a different plan for the initial step causing the side effect; and **ADJUNCT-PLAN**, which suggests making use of an adjunct plan to disable the side effect. In this example, CHEF can only find one possible instantiation, a specialization of the strategy **SPLIT-AND-REFORM**, so it is applied directly to the problem. The other modifications suggested have no actual instantiations, given the facts of the situation.
There are two stages to this process: Finding the different changes the strategies suggest, and then actually implementing the one CHEF judges to be the best in this situation.

The first step is implemented using two different techniques. CHEF first tries to find specific versions of its strategies in a library of successful previously modifications. If this fails, it uses a STRIPS-like problem solver to plan for the state suggested by the strategy.

Applying TOP—SIDE-EFFECT:DISABLED-CONDITION:CONCURRENT to failure it is not the case that: The broccoli is now crisp. In recipe BAD-BEEF-AND-BROCCOLI

Asking questions needed for evaluating strategy:
SPLIT-AND-REFORM

Asking—Can plan

Stir fry the sugar, soy sauce, rice wine, garlic corn starch, broccoli and beef for three minutes.
be split and rejoined

Found plan: Instead of doing step: Stir fry the sugar, soy sauce, rice wine, garlic, corn starch, broccoli and beef for three minutes

do:
S1 = Stir fry the broccoli for three minutes.
S2 = Remove the broccoli from the result of action S1.
S3 = Stir fry the sugar, soy sauce, rice wine, garlic, corn starch and beef for three minutes.
S4 = Add the result of action S2 to the result of action S3.
S5 = Stir fry the result of action S4 for a half minute.

Asking questions needed for evaluating strategy:
ALTER-PLAN:SIDE-EFFECT

ASKING—Is there an alternative to

Stir fry the sugar, soy sauce, rice wine, garlic corn starch, broccoli and beef for three minutes.
that will enable
The dish now tastes savory.
which does not cause
There is thin liquid in the pan from the beef equaling 4.8 teaspoons.

No alternate plan found

Asking questions needed for evaluating strategy: ADJUNCT-PLAN

ASKING—Is there an adjunct plan that will disable
There is thin liquid in the pan from the beef equaling 4.8 teaspoons.
that can be run with
S: stir fry the sugar, soy sauce, rice wine, garlic, corn starch, broccoli and beef for three minutes.

No adjunct plan found

Deciding between modification plans suggested by strategies:
Only one modification can be implemented—SPLIT-AND-REFORM

Implementing plan—Instead of doing step: Stir fry the sugar, soy sauce, rice wine, garlic, corn starch, broccoli and beef for three minutes.

do: S1 = Stir fry the broccoli for three minutes.
S2 = Remove the broccoli from the result of action S1.
S3 = Stir fry the sugar, soy sauce, rice wine, garlic, corn starch and beef for three minutes.
S4 = Add the result of action S2 for the result of action S3.
S5 = Stir fry the result of action S4 for a half minute.

Suggested by strategy SPLIT-AND-REFORM.

In this example, CHEF can only implement one of its general strategies, so the choice between possible repairs is trivial. In general, this decision is made using domain-level heuristics that determine an ordering on the specific changes CHEF is able to make to faulty plans.

CHEF tries to implement the strategies suggested by the TOP by searching for the actual plan modifications the strategies define.

7.4 Anticipating Failures
After changing the plan to solve immediate problems, CHEF also changes its understanding of the world so that it will be able to anticipate and thus avoid similar problems in the future. To do this, CHEF must figure out what it was that caused the failure in the first place. It must decide which features in the initial situation contributed to the failure so that later requests for similar goals can be planned for. Here, CHEF again makes use of the causal explanation it has for the failure to chain backward to the features that can be used to predict this kind of problem. These features are then generalized to the highest level of description allowed by the rules that explain the situation. This means taking a feature like BEEF and generalizing it to MEAT, because the rule that explains the liquid in the pan states that stir-frying any meat will have this result, so the feature can be generalized up to this level. The resulting descriptions are marked as predictive of this problem. Once this is done, CHEF is able to predict and plan for a problem of this kind whenever it encounters a similar situation.
Building demons to anticipate failure.

Building demon: DEMON0 to anticipate interaction between rules:
"Meat sweats when it is stir-fried."
"Stir-frying in too much liquid makes vegetables soggy."

Indexing demon: DEMON0 under item: MEAT by test:
   Is the item a MEAT.

Indexing demon: DEMON0 under item: VEGETABLE by test:
   Is the item a VEGETABLE.
and Is the TEXTURE of item CRISP.

Goal to be activated = Avoid failure of type SIDE-EFFECT:DISABLED-CONDITION:CONCURRENT exemplified by the failure 'The broccoli is now soggy' in recipe BEEF-AND-BROCCOLI.

Building demon: DEMON1 to anticipate interaction between rules:
"Liquids make things wet."
"Stir-frying in too much liquid makes vegetables soggy."

Indexing demon: DEMON1 under item: SPICE by test:
   Is the TEXTURE of item LIQUID.

Indexing demon: DEMON1 under item: VEGETABLE by test:
   Is the item a VEGETABLE
and Is the TEXTURE of item CRISP.

Goal to be activated = Avoid failure of type SIDE-EFFECT:DISABLED-CONDITION:CONCURRENT exemplified by the failure 'The broccoli is now soggy' in recipe BEEF-AND-BROCCOLI.

The demons associated with a failure are indexed by the surface features that are explained as having caused that failure. But these demons are also indexed by the states included as links in the overall causal chain. Liquid in the pan, for example, was the actual cause of the broccoli wilting, but was itself a by-product of the stir-frying of the meat. CHEF does not waste a piece of reasoning. Once it determines the causes of a problem, whether initial or proximate, it indexes its memory of the problem in terms of those causes. This allows CHEF greater flexibility than would indexing its recollection of a failure only in terms of the surface features associated with it. It is able to anticipate a wider range of problems without sacrificing its ties to experience.
CHEF uses its explanation of why a failure has occurred to build links that will allow it to predict the failure later, in similar circumstances.

Once a plan is debugged and the sources of any problem are marked as predictive of that problem, CHEF stores the plan using two types of features. The first is the standard vocabulary of goals met by the plan, such as what foods and tastes are included in it, as well as what type of dish it is. These features are generalized so that plans which only partially satisfy a set of goal requests, can be accessed in the absence of exact matches. The second is a different sort of vocabulary relating more to the failures encountered in building the plan than the goals the finished plan now directly satisfies. This means that the plan is indexed by the possible problems between substeps that it avoids. By storing plans this way, it is possible for the planner to use the prediction of a previously encountered problem to find a plan to solve that problem. CHEF can avoid past errors by predicting them and then finding a plan to deal with them. In the case of the BEEF-AND-BROCCOLI plan, the plan is indexed by the fact that it deals with the interaction between the meat sweating and the need to have a dry pan while cooking crisp vegetables.

Before searching for a plan at all, CHEF tries to anticipate any failures that will result from the goals it has been given, using the links formed between goal features and the failures they have caused in past episodes. By doing this before searching for a plan, the anticipation of a problem can tell CHEF to find a plan to avoid it.

The program uses its new knowledge of what went wrong in the making of the BEEF-AND-BROCCOLI recipe in order to anticipate and avoid the same problem in making a later one. Because the past failure is indexed by the features that originally caused it, CHEF is able to recall the past failure when those features arise again.

While planning for a stir-fried dish with chicken and snow peas, it predicts the possibility of the snow peas getting soggy if cooked with the chicken. It will later use this prediction to find the BEEF-AND-BROCCOLI plan which deals with this problem. This plan, rather than one having to do with either chicken or snow peas, is then modified to account for the specific goals.

Searching for plan that satisfies —
Include chicken in the dish.
Include snow pea in the dish.
Make a stir-fry dish.
Collecting and activating tests.
Fired: Is the dish STYLE-STIR-FRY.
Fired: Is the item a MEAT.
Fired: Is the item a VEGETABLE.
   Is the TEXTURE of item CRISP.
Chicken + Snow Pea = Stir frying = Failure
   "Meat sweats when it is stir-fried."
   "Stir-frying in too much liquid makes vegetables soggy."
   Reminded of BEEF-AND-BROCCOLI
   Fired demon: DEMON0
Based on features found in items: snow pea, chicken and stir fry
adding goal: Avoid failure of type
SIDE-EFFECT:DISABLED-CONDITION:CONCURRENT exemplified by
the failure 'The broccoli is now soggy' in recipe
BEEF-AND-BROCCOLI.

CHEF uses its understanding of past failures to anticipate problems
that will arise in new situations.

Once a problem has been anticipated, CHEF can use the prediction of
the problem to find a plan avoiding it. It does this explicitly by adding a
goal to avoid the problem to the list of goals it is using to search for a plan.
The BEEF-AND-BROCCOLI plan is indexed in memory by the fact that it
avoids the same problem of the interaction between meat and vegetable that
has been predicted. So CHEF is able to use the anticipation of a problem to
find the plan to avoid it.

Searching for plan that satisfies —
   Include chicken in the dish.
   Include snow pea in the dish.
   Make a stir-fry dish.
   Avoid failure of type SIDE-EFFECT:DISABLED-CONDITION:
      'CONCURRENT
   exemplified by the failure 'The broccoli is now soggy' in recipe
   BEEF-AND-BROCCOLI.
Found recipe]REC9 BEEF-AND-BROCCOLI
Recipe exactly satisfies goals] —
   Avoid failure of type SIDE-EFFECT:DISABLED-CONDITION:
      CONCURRENT
   exemplified by the failure 'The broccoli is now soggy' in recipe
BEEF-AND-BROCCOLI.
Make a stir-fry dish.

Recipe partially matches—
Include chicken in the dish.
in that the recipe satisfies: Include meat in the dish.

Recipe partially matches—
Include snow pea in the dish.
in that the snow pea can be substituted for the broccoli

When CHEF decides to use the BEEF-AND-BROCCOLI recipe as its starting point, it is passing up other recipes in memory with similarities to the current goal situation. It has, in fact, a recipe with chicken and green beans that, in the absence of the BEEF-AND-BROCCOLI experience, it would have been happy to modify to account for the chicken and snow peas. But because it has encountered a failure stemming from a similar situation in the past, it anticipates the problem and prefers a plan for the situation that we built to deal with that problem. It favors this plan, although it has fewer surface features in common with the current situation than another plan in memory, because it knows this plan deals with an interaction between goals not addressed by the other. This interaction between plan steps, because it has so great an effect on the final structure of the plan, is more important to CHEF that the details of the surface goals in choosing its initial plan.

CHEF uses its anticipation of a problem to find a plan to avoid it.

Once this plan is retrieved, CHEF then makes the changes needed to fit it to the current goals,

Building new name for copy of BEEF-AND-BROCCOLI based on its goals.
Calling recipe CHICKEN-STIR-FRIED

Modifying recipe: CHICKEN-STIR-FRIED
to satisfy: Include chicken in the dish.
Placing some chicken in recipe CHICKEN-STIR-FRIED

Modifying recipe: CHICKEN-STIR-FRIED
to satisfy: Include snow pea in the dish.
Placing some snow pea in recipe CHICKEN-STIR-FRIED
and thus produces the required recipe:

Created recipe CHICKEN-STIR-FRIED

CHICKEN-STIR-FRIED
A half pound of chicken
Two tablespoons of soy sauce
One teaspoon of rice wine
A half tablespoon of corn starch
One teaspoon of sugar
A half pound of snow pea
One teaspoon of salt
One chunk of garlic

Bone the chicken.
Chop the garlic into pieces the size of matchheads.
Shred the chicken.
Marinate the chicken in the garlic, sugar, corn starch, rice wine and soy sauce.
Chop the snow pea into pieces the size of chunks.
Stir fry the snow pea for three minutes.
Remove the snow pea from the pan.
Stir fry the spices, rice wine and chicken for three minutes.
Add the snow pea to the spices, rice wine and chicken
Stir fry the spices, snow pea, rice wine and chicken for a half minute.
Add the salt to the spices, snow pea, rice wine and chicken.

Surface features of a situation in which a failure has been anticipated are important, however. If CHEF were given the goals to make a stir-fried dish with chicken and bean sprouts after storing its new CHICKEN-STIR-FRIED plan in memory, it would anticipate the problem with the bean sprouts getting soggy, but would not use the BEEF-AND-BROCCOLI plan as its baseline. Instead, it would use the CHICKEN-STIR-FRIED plan that deals with the problem of stir-frying meats and crisp vegetables together and has more surface features in common with the current situation than the other plan that also dealt with this problem, the BEEF-AND-BROCCOLI recipe. Once a failure is anticipated, it becomes one of the features used to plan, but not the only one.

CHEF is a model of memory-intensive planning. CHEF designs new plans out of its memories of old ones. It uses memories of successful plans as the basis for new ones. It uses memories of its own failures to warn itself of possible problems. And it uses memories of past repairs to fix those problems when they arise.

As CHEF plan, it learns. It learns new plans by creating successful recipes and storing them in memory. It learns to predict problems by experiencing
and explaining failures. It also learns new ways to avoid the problems it predicts by repairing failed plans and storing the repairs for later use. The essential idea of CHEF is that planning is interaction with the world and that interaction can lead to learning.

8. LEARNING PLANS

A case-based planner learns plans in the sense that it stores the new plans it creates in its plan memory. Learning, in this context, means figuring out the features that should be used to index the plan in memory, and then storing the plan using them. The task is to organize experience so that it can be retrieved and reused at the appropriate time.

A case-based planner stores the most obvious result of planning, the plan itself, in a plan memory.

The object that a case-based planner stores in memory is the plan itself. The plan is a series of steps and a list of ingredients that has been built to satisfy some particular set of goals. This plan is not generalized in any way. To do so would be to lose information that could be used again, without gaining anything in terms of more general applicability.

For example, in the CHEF program, altering a specific plan such as STRAWBERRY-SOUFFLE, in order to satisfy a slightly different set of goals, such as including kirsch, is no more difficult than altering a generalized version of the same plan. But in a situation where the planner has to plan for goals identical to those in the original situation, a generalized plan would be more difficult to adapt, because the specifics of the original plan would be lost.

While a case-based planner need not generalize the plans it stores, it does need to generalize the features used to index them. Because of this, the plan can be suggested for use in a wide range of situations and can still retain the specific information that makes it more useful when applied to situations in which the goals that are being planned for are completely satisfied by the plan itself. By generalizing the indices rather than the plan, a case-based planner is able to avoid much of the trade-off between generality and power of application. The general indices make it applicable in many situations and the specificity of the plan makes it a powerful tool in those situations in which the match between the current goals and those satisfied by the plan is a good one.

Because all indices are generalized (e.g., BROCCOLI to VEGETABLE BEEF to MEAT) the task of retrieval replaces that of matching. These are used to place the plan in a discrimination net that is later searched by the RETRIEVER. When a feature used in retrieval has to be abstracted (can
not find BROCCOLI, look for VEGETABLE), a further discrimination is
done using subfeatures of the original (once down VEGETABLE, discrim-
inating further TEXTURE = CRISP). Because a strict hierarchy of goals is
assumed (corresponding to features), search of the discrimination net is
always downward-driven with no backup.

In storing a plan in memory, a case-based planner does not generalize
the plan itself. Instead, it generalizes the features used to index the
plan in memory.

In the CHEF program, the goals include the taste and texture of the dif-
erent items in a plan and the ingredients included in it. These goals, along
with the goal to make the type of dish that the plan builds, are the initial
features used in indexing the plan. Problems a plan avoids are represented
by the TOP describing the abstract causal situation linked to the specific
failure state. For example, the BEEF-AND-BROCCOLI plan, which was
repaired to avoid the problem of soggy vegetables, is marked as dealing with
this problem. In CHEF, a plan can have many problems due to goal interac-
tions that it solves, just as it can satisfy many goals directly. These two sets
of items are combined into a single list of features that will be used to index
the plan in memory.

The fact that a plan avoids a particular problem can be treated by a case-
based planner as a goal to avoid the failure that the plan associated with it
satisfies. These goals to avoid failures can be treated as any other goals for
use in indexing of plans in memory. They may be considered to be more im-
portant than most other goals, however, because it is often easier to alter a
plan to satisfy a new goal than to make the changes required to deal with an
interaction among goals.

A plan is indexed in memory by the goals it satisfies and the problems it
avoids. It also has to be indexed by the features of a situation that are in-
dependent of the goals but do direct the planner to one plan or another. As
mentioned before, the plan to call the front desk for a wake-up call is a
good plan to get the planner up in the morning, but only in certain circum-
stances. A description of these circumstances must be part of the scheme for
indexing plans in memory.

In the CHEF program, the actual implementation of its memory of plans
is via a discrimination net in which plans are indexed by the goals they satisfy
and the problems they avoid. These goals and problems are ordered by their
importance, with the higher priority features used at the higher levels of dis-
crimination. Once the features are ordered, a plan is placed in the discrim-
ination net, using the features as indices. The ordering of features allows
CHEF to limit somewhat the branching in this net by allowing discrimination
from any one node to be made only on the basis of features of less impor-
tance than the last one used to index to it.
Created recipe STRAWBERRY-SOUFFLE

If this plan is successful, the following should be true:

- The batter is now baked.
- The batter is now risen.
- The dish now tastes like berries.
- The dish now tastes sweet.

The plan satisfies —
- Include strawberry in the dish.
- Make a souffle.

Figure 7. Goals satisfied by STRAWBERRY-SOUFFLE.

As each goal is used to place a plan in memory, more general versions of the goals are also used. BEEF-AND-BROCCOLI is indexed by the fact that it includes beef and broccoli, but it is also indexed by the fact that it includes meat and vegetables in general. The level of generality of the goals used to index plans is preset and corresponds to the level of generality used in defining the modification rules used in substituting one ingredient for another in a plan. As a result, the goal to include one type of ingredient can be used to find a plan for a similar ingredient.

CHEF stores new plans indexed by the goals they satisfy and the problems they avoid.

Once a plan is placed in memory, it can be accessed by the planner when it is searching for a plan to satisfy similar goals while avoiding failures it predicts may appear.

For example, in building STRAWBERRY-SOUFFLE recipe, CHEF has to deal with the fact that the liquid from the fruit disables the conditions required for the soufflé to rise. It deals with this by adding more egg white to the recipe, thus reestablishing the relationship between liquid and leavening the fruit put out of balance. The final recipe satisfies a set of goals having to do with making a soufflé, including strawberries, making the soufflé taste sweet and like berries, and having a fluffy texture. These goals come from the input request, and some are generated by the planner out of its understanding of what this type of plan should be like in general (Figure 7). The plan also has a token associated with it indicating that it deals with the problem of the added liquid from the fruit interfering with the baking (Figure 8, p. 430).

These goals are collected and used to index the plan in memory, the goal to make a soufflé and the goal to avoid the failure taking priority over all others. These are then used to insert the plan into CHEF's plan memory. The only generalization that goes on is with the goal to include the strawberries, which is generalized up to the level of FRUIT. There are no other goals that can be further generalized.
Created recipe STRAWBERRY-SOUFFLE

The plan avoids the failure of type SIDE-EFFECT:DISABLED-CONDITION:BALANCE exemplified by the failure 'The batter is now flat' in recipe STRAWBERRY-SOUFFLE.

Figure 8. Avoidance goal under STRAWBERRY-SOUFFLE.

Once stored, this plan can be found whenever the planner is trying to plan for any subset of the goals it satisfies and can be found in any situation in which the problem of an imbalance between liquid and leavening is predicted. Even without the prediction of the failure having to do with the liquid from the fruit giving the plan problems, it could be found when the planner is given the goals to make a soufflé with fruit. Because it is also indexed by the fact that it does avoid this particular problem with added liquid, it can also be accessed when the planner is given the goals to make a kirsch soufflé, which activates the prediction of the problem, and then allows CHEF to find the plan. So, this additional index allows the planner to find the plan in situations not matching the individual goals, but having problems occurring in them the plan solves.

9. LEARNING TO PREDICT FAILURES

A case-based planner can use its own memories of past failures to anticipate new ones by associating failures with the surface features predicting them. This association can be made at the time of a failure by forming links between features of goals and the failures they have caused. These links can then be used to activate the memory of the failure when the goals arise again, thus giving the planner the warning that it should find a plan to avoid the problem. The problem of learning to predict failures in order to anticipate and avoid them, is a problem of figuring out which features of an existing situation have caused a current failure, so that these features can be linked to the memory of the failure.

A case-based planner saves memories of failures, indexed by the features predicting them. It uses these memories to anticipate problems when these features arise again in new situations.

When reminded of a failure, the planner needs to recall the token built to represent the failure, and to index the plan that avoids it in memory. By having the planner be reminded of the same representation of the failure
used to index the plan which avoids it in memory, it can use the reminding of the failure directly in searching for the appropriate plan.

For example, CHEF's STRAWBERRY-SOUFFLE plan is indexed by the fact that it avoided a particular problem. The token used to index the plan is then associated with the planner's memory of the failure. Because of this, the activation of the memory of the failure could be used directly to index to the plan dealing with it.

In saving a memory of a failure, CHEF saves the token used to index the plan that avoids that failure in CHEF's plan memory.

A case-based planner that does any sort of causal analysis of a problem in order to repair it has a powerful tool for figuring out which features in a situation should get the blame for causing a problem. The explanation created to fix a faulty plan can also be used to identify the features of a situation that will later predict that fault. As in learning new plans, the planner does most of the work and the learner's task is to form the links that will recall that work. The planning task is to repair the plan by figuring out what went wrong and then changing the circumstances that caused the problem. The learning task is to recall these circumstances so the planner can recognize when a failure is imminent and search for a plan to avoid it.

CHEF, for example, uses the explanation of why a failure has occurred to point out which features of the current situation are responsible for the problem and then uses it again to find the features that will be predictive of the problem in later situations. These are the same features, but the learner in CHEF wants to know not only the exact features that caused the problem, but also the more general versions of them that might cause it again. It gets these more general features by doing what I call generalizing to the level of the rules. This means generalizing an object in an explanation to the highest possible level of generality while still staying within the confines of the rules that explain the failure.

The difference between this notion of explanation and that suggested by other theories of explanation-driven learning (e.g., DeJong & Mooney, 1986; Mitchell et al., 1986) lies in what is learned. Within case-based planning, the explanation of why a plan has failed is used to figure out the features that will later predict similar failures. In other theories of explanation-driven learning, explanations of why a plan has been successful are used to weed out irrelevant steps and generalize those that are too specific. Explanation of failures is a far more constrained task in that only a single anomalous state has to be accounted for and the planner is not trying to generalize the plan it is building at all; it is only trying to generalize its understanding of the circumstances in which that plan can be used.
Building demons to anticipate failure.
Building demon: DEMON2 to anticipate interaction between rules:
"Chopping fruits produces liquid."
"Without a balance between liquids and leavening the batter will fall."
Indexing demon: DEMON2 under item: FRUIT
by test: Is the item a FRUIT.
Indexing demon: DEMON2 under style: SOUFFLE
Goal to be activated=Avoid failure of type SIDE-EFFECT:DISABLED-CONDITION:BALANCE exemplified by the failure 'The batter is now flat'
in recipe STRAWBERRY-SOUFFLE.

Figure 9. Marking FRUIT and SOUFFLE and predictive of problems.

A case-based planner uses the explanation of a failure to identify the features that will predict it. It generalizes these features to the highest level of description allowed by the rules in the explanation.

In an example of the CHEF program dealing with the problem of a failed strawberry soufflé, in which the liquid from chopped strawberries causes the soufflé to fall, one of the links in the chain of the explanation is provided by a rule stating that the extra liquid in the batter was a product of chopping the strawberries. A simple way to avoid this failure in the future would be for the planner to mark strawberries as predictive of it, and be reminded of the failure and the repaired plan whenever it is asked to make a strawberry soufflé. But it turns out that the rule explaining the added liquid as a side effect of chopping the strawberries, does not require that the thing chopped be strawberries at all. In fact, it explains that chopping any fruit will produce this side effect. Instead of marking STRAWBERRY as predictive of the problem, CHEF can mark FRUIT as predictive (Figure 9).

In some cases, the rule explaining a link in the causal chain leading to a failure does have specific requirements. For example, in explaining the soggy broccoli in the BEEF-AND-BROCCOLI plan, CHEF uses a rule explaining that stir-frying any crisp vegetable in liquid will make it soggy. In this case, BROCCOLI is too specific a feature to hang the prediction from, and VEGETABLE is too general. To deal with this, CHEF uses the tests on the rule itself to control the activation of the failure prediction. These tests are associated with the general ingredient type that the rule tests for, VEGETABLE, and tests for the features required for the rule to apply (TEXTURE = CRISP). This means that each time the planner has to deal with stir-frying vegetables, it will test their texture and partially activate the
memory of the failure if they are crisp. These tests are always checks for existing semantic features. As a result, they can take the form of a network through which markers can be passed without the need for explicit inference. This means that anticipating known failures is a low-cost task.

In most situations, it is not a single feature, but a set of features combining to cause a failure. The BEEF-AND-BROCCOLI situation is one of these because the failure is not the result of either stir-frying the beef or stir-frying the broccoli, but is the result of stir-frying them together. In these situations, a single feature alone should not activate the prediction of the failure. It is only when all of the features are together (the style of the dish is STIR-FRY, one ingredient is MEAT and another ingredient is a VEGETABLE that has TEXTURE = CRISP), that the prediction should be made. To deal with this, the activation of a single feature linked to a failure only partially activates the memory. It is only when all these features send signals that they are present that the memory of the failure itself is activated, and the planner is warned of the impending problem. CHEF does not anticipate a problem when it plans for crisp vegetables alone. It only does so when it is asked to plan for stir-frying crisp vegetables along with other goals that will combine with the first to cause the problem again.

When multiple features are required to predict a failure, all of them are linked to the memory of the failure. This memory is not activated unless all of the linked features are present.

Tracing through the explanation of a failure to initial causes, a planner passes through states those original causes have created. These states are the more proximate causes of the problems, but not the first causes in terms of the recipe. The liquid from chopping the strawberries is the actual cause of the problem with the strawberry soufflé, but the initial cause is the goal to include strawberries itself.

Although these states are not the initial causes of the problems the planner has to deal with, they can still be used to predict the problems in later situations. A planner has to handle these intermediate states in the same way it handles the ingredients it has to mark as predictive of problems. It generalizes them to the level of the rules explaining the failure and links them to the token representing the failure. If other conditions are also required for the failure to occur, they are also linked to the memory of the failure, so that one feature alone will not predict the failure when it is inappropriate.

The CHEF program responds to the failure of the strawberry soufflé by marking the goal to include any liquid spice in a soufflé as predictive of that soufflé falling (Figure 10, p. 434). This is implemented by placing a test on the concept SPICE, which checks for the texture, and partially activates the memory of the failure when the test is true.
Building demons to anticipate failure.

Building demon: DEMON3 to anticipate interaction between rules:
"Liquids make things wet."
"Without a balance between liquids and leavening the batter will fall."

Indexing demon: DEMON3 under item: SPICE
by test: Is the TEXTURE of item LIQUID.

Indexing demon: DEMON3 under style: SOUFFLE
Goal to be activated=Avoid failure of type
SIDE-EFFECT:DISABLED-CONDITION:BALANCE
exemplified by the failure 'The batter is now flat'
in recipe STRAWBERRY-SOUFFLE.

Intermediate states serving as links in the causal chain leading to a failure are also linked to the memory of the failure and thus can be used to predict it if they arise in a later situation.

When a set of features combine to predict a failure, a case-based planner is reminded of the memory of the failure itself. In CHEF, the request for a soufflé with raspberries in it reminds the planner of the problem of extra liquid in a soufflé and allows it to find and modify the STRAWBERRY-SOUFFLE plan dealing with it. It also allows CHEF to be reminded of it when planning for a kirsch soufflé, in which the added liqueur would have the same effect as the liquid from the chopped fruit. This is a plan the planner would not have used had it not been for the prediction of the failure. The fact that the ASSIGNER has previously established these links between features and failures allows it to find and make use of plans that otherwise would have not interesting features in common.

Before closing the discussion of learning to predict failures, there is one point that must be made about indexing in a case-based planner. A case-based planner indexes its plans by descriptions of the problems they avoid. It also indexes memories of these problems by the features predicting them. Given this, the question arises: Why not just index the plans by the features predicting the problem they solve directly?

There are two answers to this question. First, because a single class of problems can have many specific causes, it is simply more efficient to index the plan under a single characterization of the problem and have the problem be predicted independently, instead of indexing the plan by all possible circumstances that might cause the problem. Problems can be predicted with only a few features. There is little interaction between them that would lead to the need for a complex indexing system. But adding these features to the
those already used to index plans would increase the complexity of plan indexing dramatically.

Second, efficiency is not the only argument. A more compelling argument is that indexing plans that solve problems by the features predicting them, without marking those features in any way, does not work. As was pointed out earlier, it is often the case that the best plan for a situation is not the one that has the closest match in terms of low-level features.

In CHEF, for example, the prediction of the "soggy vegetable” failure allowed the program to find the BEEF-AND-BROCCOLI plan that avoided it. Without this prediction, however, the planner would have to find a plan using only the features of the initial goals. Because the planner does have a plan for CHICKEN-AND-PEANUTS in memory, this would be taken as the baseline plan. But this plan, when modified for the current goals, will lead to soggy snow peas while a modified BEEF-AND-BROCCOLI will not.

The fact that a case-based planner indexes plans by the problems they solve and then predicts these problems for use in search, allows it to recover them in situations where the low-level features of the current goals would lead it to less appropriate plans.

10. LEARNING CRITICS

Aside from storing plans and failures, a case-based planner also stores some of the repairs it makes to plans so that they can be used again. These repairs, stored in the form of critics, allow a planner to repair plans it knows to be faulty before it runs them. A planner would run into this situation when it predicts a problem, but cannot find a plan of the proper type to deal with that problem. When this occurs, it has to use a plan it knows to be faulty and then change it with the same patch it used to repair another faulty plan in the past.

A case-based planner stores some of the repairs it makes, indexed by the problems they solve

As with learning plans, the task of the learner is not to build the repair that is going to be stored, but only to decide how it is going to be indexed, so that it can be accessed at the right time. There is a difference, however, in that not all repairs can be saved. Some repairs, because they involve interactions among many parts of a plan, are too complex to transfer to new problems. Others are also linked to the type of plan being built, so there is never a possibility that the problem is predicted but no plan of the dish-type needed can be found. Aside from deciding where to store the repair, the learner also has to decide which repairs can be saved at all.
The decision to save a repair as a critic is based on the repair strategy used to build the specific repair, and on the cause of the failure in the first place. Some failures are the products of multiple ingredients interacting, so no one ingredient can be blamed for the problem, and thus, no one ingredient can be given a critic to perform the repair. The first test for turning a repair into a critic is whether it relates to a single ingredient. The second test depends on the complexity of the repair itself. Some strategies, such as ALTER-PLAN:PRECONDITION, ALTER-PLAN:SIDE-EFFECT and SPLIT-AND-REFORM depend on specific steps being present in a plan, and their changes cannot be reused reliably. On the other hand, strategies such as REORDER and REMOVE-FEATURE create simple changes that can be added to most plans. For example, the addition of a step to remove the fat from duck, created by REMOVE-FEATURE, can be added to any plan involving duck. Likewise, the ordering change suggested by REORDER when the marinating of shrimp blocked shelling it later, can be applied to any case of the two steps being misordered.

A case-based planner can only save those repairs that can be transferred to any plan in which the problem they repair arises.

In turning a repair into critic, a case-based planner has to store the specific change suggested by the strategy under the ingredient it relates to. Again, this is so the prediction of a problem can lead to finding the repair that will fix it. By doing this, the past repair can be suggested when, and only when, the problem it relates to is noticed or predicted.

In CHEF, for example, a repair that adds the step of removing the fat from the duck is stored under the concept DUCK itself. Unlike other repairs that take the form of critics, however, new critics are indexed by the problems they solve. To activate them, CHEF has to anticipate the problem and then fail to find a plan solving it. This restriction prevents CHEF from applying a repair to a plan that has already been fixed. It only applies the new critic when the features predicting the problem are anticipated and the plan used by the planner fails to deal with it.

Building a critic out of a repair requires only that the planner put together the pieces that have already been built or identified. The explanation that is used to repair the plan in the first place identifies where a critic should be stored. The critic itself is the actual repair built by the strategy to patch the plan in the first place.

CHEF builds new critics out of current repairs and then indexes them by the description of the problem being repaired.
Applying TOP—SIDE-FEATURE:GOAL-VIOLATION

to failure The bunch of dumplings is now greasy.
in recipe BAD-DUCK-DUMPLINGS

Implementing plan—After doing step: Bone the duck
do: Clean the fat from the duck.
Suggested by strategy REMOVE-FEATURE

Figure 11. Repairing DUCK-DUMPLINGS with REMOVE-FEATURE.

Building critic to avoid failure: The bunch of dumplings is now greasy.
caused by condition: The duck is now fatty.

Critic=
After doing step: Bone the duck
do: Clean the fat from the duck
because: The duck is now fatty.
Storing critic under DUCK
Indexing by SIDE-FEATURE:GOAL-VIOLATION0

Figure 12. Storing new critic under DUCK.

Problem predicted — SIDE-FEATURE:GOAL-VIOLATION0

Considering critic:
After doing step: Bone the duck
do: Clean the fat from the duck
because: The duck is now fatty. — Critic applied.

Figure 13. Using a new DUCK critic.

An example of this notion when the CHEF program has to confront the problem of the grease from the duck fat. In dealing with the fat from the duck making the dish greasy, CHEF repairs the plan using the strategy REMOVE-FEATURE (Figure 11). Because the repair is a simple one having to do with only one ingredient, it can then go on to store it as a critic under DUCK, indexed so that the prediction of the problem will activate the critic for use in a different plan (Figure 12). When it later has to make a duck pasta dish and cannot use its DUCK-DUMPLINGS, it is in the position of predicting a problem that it does not solve with a complete plan. During modification, it activates the critic and adds the step removing the fat from the duck (Figure 13). Even though it could not find a complete plan to deal with the problem then, it is able to avoid it before running its new plan by using its earlier repair.

As with the other aspects of a case-based planner’s learning, the stress in learning new critics is not on the construction of the critics themselves. The stress is on how the critics are to be stored so that they can be accessed at the
appropriate time. By indexing the patches to a plan under the prediction that it will fail, a case-based planner can activate those patches only when the features predicting the failure are present.

11. CASE-BASED REASONING

CHEF is part of a growing movement in artificial intelligence involving the use of memory in planning (e.g., Alterman, 1985; Carbonell, 1981, 1983; Hammond, 1983; Kolodner & Simpson, 1984; Kolodner, Simpson, & Sycara, 1985), problem solving (e.g., Rissland & Ashley, 1986; Simpson, 1985), language understanding (Martin & Riesbeck, 1986) and diagnostic systems (Hammond, 1987; Hammond & Hurwitz, 1988; Koton, 1988). The main feature linking all of these systems together is that the reasoning they perform is driven by an episodic memory rather than a base of inference rules or plan operators.

Like CHEF, the case-based systems PLEXUS (Alterman, 1985) and MEDIATOR (Kolodner et al., 1985) have tried to address issues of planning from a dynamic memory. PLEXUS's memory was designed to provide nearly right plans for an execution-time improvisational system. MEDIATOR, on the other hand, used plans pulled from memory as starting points to guide new reasoning. Together, CHEF, PLEXUS, and MEDIATOR cover much of the range of planning problems occurring in the world. PLEXUS deals with those execution-time problems that can be solved by directly applying an existing plan selected from memory on the basis of environmental cues. MEDIATOR is aimed more at problems arising from classification errors that can be solved by building new categories in memory. CHEF, on the other hand, is aimed at problems of interaction between planning steps discovered during plan execution. Like CHEF's, MEDIATOR's memory is changed as a result of its experience.

The major difference between CHEF and PLEXUS lies in the use of causal knowledge. In CHEF, all repair is based on the use of a deep domain model that provides explanations of the failures that occur. This gives CHEF the ability to repair a wide range of problems correctly, but limits it to those problems it has the ability to explain. PLEXUS, on the other hand, does not require a causal model, in that it repairs plans by selecting alternative subplans from memory on the basis of environmental features. This frees it from the need for an explicit domain model, but also limits it in terms of both correctness and range of possible repairs. In CHEF, the analog to the improvisation in PLEXUS is the use of object critics to tweak an almost-right plan. It is clear that both of these approaches are needed in order to create a functioning case-based planner.

The major difference between CHEF and MEDIATOR is vocabulary. One of the primary issues in CHEF is the use of abstract descriptions of
planning problems in indexing. In CHEF, solutions in the form of general strategies, as well as specific plans, are indexed using a vocabulary of goal and plan interaction. In MEDIATOR, the vocabulary is more closely linked to the features of the domains in which it operates. Here the trade-off is again between the overhead involved with inference needed to apply the more abstract vocabulary, and the power gained through its use. As with PLEXUS, the relationship between CHEF and MEDIATOR is a complementary one. Well-understood problems can be handled faster using the vocabulary in MEDIATOR, while newer problems require the use of a domain model. The analog to the MEDIATOR vocabulary in CHEF is the network of links among surface features, and memories of existing problems used to anticipate those problems when those features are present.

Another line of research in case-based reasoning, which has shown a great deal of promise, has been in the area of legal reasoning (Rissland & Ashley, 1986). This work has centered around the use of cases, in the legal sense of the word, to support decision making and argument. Like the CHEF research, much of the thrust in this work has been aimed at the issue of representation. In particular, it has involved the use of a strategic approach to the alteration of input features in order to find relevant cases in memory. This use of strategic modification has resulted in systems that can retrieve cases from memory that are similar, but strategically different than an initial input case. By finding these cases that differ along specific dimensions, the system can reinforce its own arguments, as well as anticipate problems an adversary may throw in its way. The most exciting aspect of this work is the use of explicit strategies in the selection of features for use in indexing. This strategic approach has been all but ignored in other case-based systems, yet is clearly an important aspect of human reasoning (Seifert, McKoon, Abelson, & Ratcliff, 1985).

Along with the generative work in planning and problem solving, the case-based approach has been successfully used in both language understanding and diagnosis. In their DMAP work on using memory in parsing, Martin and Riesbeck (1986) have advanced the idea of parsing directly into a dynamic memory by passing markers up from lexical items to concepts in memory. This activation results in predications also being passed back through memory to particular lexical items. The overall approach involves treating understanding primarily as a recognition task in which all aspects of understanding (i.e., parsing, inference, and disambiguation) are handled through the use of a single mechanism and memory. In recent work in diagnosis, Koton (1988) has proposed a case-based approach to diagnosis, which again treats the task as one of recognition. Using an existing expert system as a data source, Koton’s CASEY was able to construct a case base allowing it to perform the same task using less computational power with more graceful degradation at the edges of its knowledge.
CHEF is only part of what seems to be a very fruitful approach towards reasoning. In all of these systems, the philosophy is that the search for a solution to a new problem should begin with a solution to an old one.

12. HOW LEARNING FROM PLANNING IS DIFFERENT

Learning for a case-based planner is a by-product of planning, so the needs of the planner determine what is going to be learned, when it is going to be learned, and how it is going to be learned. Other learners, which learn categories or plans in the absence of a use for what they learn, are concerned with creating new structures, but have little interest in how they are stored and managed for use in an active memory. A learner associated with a planner, on the other hand, does not do any building on its own. Instead, it is concerned with how to organize the results, both positive and negative, of its planner in a dynamic memory of experience the planner can use.

Learning is the organization of experience. As a result, the core issues of learning are memory organization and indexing. The most important problem in learning is deciding how to describe and store an experience so that it can be used again. In a planner that learns, this means finding a way to organize the planner's results so they can be found again in the appropriate situations.

A case-based planner must predict problems so that it can find plans to avoid them; so, it makes sense for it to learn the features predicting the problems it encounters. Because it is possible to predict a problem that cannot be handled with an existing plan, it also makes sense for a planner to learn specific fixes that can be applied to repair the plan it has to use. The needs of the planner decide what is going to be learned: plans, problems, and repairs.

Planning is a test of the planner's knowledge. When that knowledge fails, the planner can see by its own experience that it needs to learn some new way to discriminate between the situation it thought it was in and the one it is actually in.

Because a planner is able to test its knowledge of the world by building new plans, its planning experiences can tell it when it has to learn something from those experiences. When a case-based planner builds a successful plan, it knows to store it for later use. Because it knows in what sense the plan is successful (in that it knows the goals it satisfies and the problems it avoids), it also knows how to index it in memory. When a case-based planner fails, it knows to mark the features that causing the failure as predictive of it so that it can anticipate it in the future. The failure itself tells the planner that its knowledge has a gap, and the gap has to be filled with the prediction of a problem the planner was unable to anticipate before. When a case-based planner repairs a plan, it knows this repair is a patch to fix a failure created by its own modification process. In response, the planner adds this patch to
the knowledge used by that process. The planner determines when the learner is turned on: when the planner succeeds, fails, and repairs.

Finally, the planner itself provides the learner with the content of what is learned. It builds the plans stored in memory. It builds the explanation that is used to assign blame for a failure. It even builds the patch stored as a new ingredient critic. The learner does not create what is learned, the planner does. In every case, the task of the learner is the task of collecting the features that should be used to index the planner's work and then store that work away for later use.

The planner does the reasoning. The learner examines that reasoning and decides how it should be stored in memory so that it can be recalled again when needed. Unlike systems that do nothing but learn, and as a result do nothing with what they learn, a case-based planner that learns is managing a dynamic memory of experience in service of planning.

Learning means storing the different results of the planner's activity indexed by the features determining their usefulness.

13. HOW LEARNING FROM PLANNING IS BETTER

A case-based planner builds functional categories, in the mold of Schank, Collins, and Hunter (1986), allowing it to anticipate problems and respond to them. This can be contrasted with programs building definitional categories, which are no more than lists of necessary and sufficient features. The notion of learning not only a plan, but the features it should be indexed by, is an improvement over other learners aimed at plan learning. A planner is concerned with the reuse of its plans. As a result, it cannot be satisfied with just having the plan in hand, it must understand when it should be used, and store it in such a way that it will be recalled at that time.

A case-based planner can use its planning knowledge to guide the credit-assignment decisions it makes in marking features as predictive of problems. This helps it to avoid the difficulties associated with inductive learning algorithms. The planner only has to see one instance of a problem in order to learn the features predicting it. It ignores extraneous features. It uses the explanation of a situation to determine the level of generality those features will be pushed to. And it does not have to rely upon a tutor to hand it the correct set examples for generalization. By using the knowledge of what it is learning to guide it, a case-based planner can replace credit assignment through repetition, with a more powerful credit assignment through causal relevance.

A planner knows when to learn. Unlike learning systems that do nothing but learn, a planner actually has a motivation for learning: the improvement of its planning abilities. Because it learns in response to the needs of
the planner, it learns when, and only when, the planner requires it to. Because it learns in order to improve the planner, its learning is constrained to be only that which will help the planner with later efforts. The stress of the learning is on the use of what is learned instead of the moment of learning itself.

Learning from planning is an improvement over other types of learning in that it uses the knowledge of the planner to determine what it learns, how to index it, and whether to learn it at all.

All of these improvements come from the basic idea that learning is not separate from planning. Learning is the management of the planner's memory of its own experience so that it can plan more effectively and avoid the problems it has encountered before. Learning does not just involve memory: Learning is memory.

REFERENCES


