CHEF: A Model of Case-based Planning.*

Kristian J. Hammond

Department of Computer Science
Yale University

ABSTRACT

Case-based planning is based on the idea that a machine planner should make use of its own past experience in developing new plans, relying on its memories instead of a base of rules. 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.

Successful plans are stored in memory, indexed by the goals they satisfy and the problems they avoid. Failures are also stored, 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.

These ideas of memory, learning and planning are implemented in the case-based planner CHEF, which creates new plans in the domain of Szechwan cooking.

I WHAT IS CASE-BASED PLANNING?

Case-based planning is planning from experience. A case-based planner differs sharply from planners that make use of libraries of goals and plans in that it relies on an episodic memory of past planning experiences. As a result, memory organization, indexing, plan modification, and learning are central issues in case-based planning. Because this sort of planner has to be able to reuse the plans that it builds, it must be able to understand and explain why the plans that it has built succeed or fail in order to properly store them for later use. This means that a cased-based planner not only needs to have a powerful memory organization, it must also have a strong model of the causality of the domain in which it operates.

The case-based approach to planning is to treat planning tasks as memory problems. Instead of building up new plans from scratch, A case-based planner recalls and modifies past plans. Instead of seeing plan failures as planning problems alone, it treats them as expectation failures that indicate a need to modify its understanding of the world. And instead of treating plans as disposable items that are used once and then forgotten, a case-based planner treats them as valuable commodities that can be stored and recalled for later use. In general, a case-based planner treats at the problem of building and maintaining plans as an interaction between its knowledge base and the world. Any problem that arises out of a disparity between what the planner knows and what is requires that it alter not only its plan but also the expectations that led to the creation of that plan.

II WHY CASE BASED?

The argument for case-based planning is straightforward: we want a planner that can learn complex plans rather than replan every time it has to achieve an already planned for set of goals. In the case of a single plan from the CHEF domain, a plan for stir fried chicken and peanuts for example, the plan itself has seventeen steps performed on a dozen ingredients. While such a plan can be built up from a set of rules or plan abstractions, it is more efficient to save the entire plan and recall it for reuse when the same situation reoccurs. Further, the case-based approach seems to much more closely reflect human planning behavior than do those approaches that suggest replanning for every new case.

Even going back to the earliest rule-based planners such as STRIPS [1], there has always been the desire to save completed plans in a way that makes them accessible to the planner in later situations. This was especially the case in those situations where a past plan included information about how to avoid problems that the planner's base of rules tended to lead into. But the planning algorithm used by most planners does not allow for anything but the most obvious reuse of existing plans. Most planners build plans for multiple goal situations out of the individual plans for each of the goals that the planner is handed and then deal with any interactions as they arise. Unfortunately, this algorithm has resulted in set of planners that recreate and then debug past mistakes rather than using the plans that they have developed before that avoid those mistakes altogether [2,4,5].

The approach taken in CHEF is to *anticipate and avoid* problems due to plan interaction. To do this, CHEF keeps track of what features in its domain are predictive of particular problems so it can predict when it has to plan for them. It also saves plans in memory, indexed by the goals that they satisfy and the problems that they avoid. So the prediction of a problem allows CHEF to find the plans in memory that avoid it. CHEF's basic algorithm is to find a past plan that satisfies as many of the most important goals as possible and then modify that plan to satisfy the other goals as well.

III CHEF'S OVERALL STRUCTURE

CHEF is a case-based planner that builds new plans out of its memory of old ones. CHEF's domain is Szechwan cooking and its task is to build new recipes on the basis of a user's requests. CHEF's input is a set of goals for different tastes, textures, ingredients and types of dishes and its output is a single recipe that satisfies all of its goals. Its output is a single plan, in the form of a recipe, that satisfies all of the users goals.

Before searching for a plan to modify, CHEF examines the goals in its input and tries to anticipate any problems that might arise while planning for them. If a failure is predicted, CHEF adds a goal to avoid the failure to its list of goals to satisfy and this new goal is also used to search for a plan. Because plans are indexed in memory by the problems they avoid, this prediction can be used to find a plan that solves the predicted problem. Much of CHEF's planning power lies in this ability to predict and thus avoid failures it has encountered before.

*This report describes work done in the Department of Computer Science at Yale University. It was supported in part by ONR Grant #N00014-85-K-0108.

CHEF consists of six processes:

- **Problem anticipation:** The planner anticipates planning problems by noticing features in the current input that have previously participated in past planning problems.
- **Plan retrieval:** The planner searches for a plan that satisfies as many of its current goals as possible while avoiding the problems that it has predicted.
- **Plan modification:** The planner alters the plans it has found to satisfy any goals from the input that are not already achieved.
- **Plan repair:** When a plan fails, the planner fixes the faulty plan by building up a causal explanation of why the failure has occurred and using it to find the different strategies for repairing it.
- **Credit assignment:** Along with repairing a failed plan, the planner wants to repair the characterization of the world that allowed it to create the failed plan in the first place. It does this by using the causal explanation of why the failure occurred to identify the features in the input that led to the problem and then mark them as predictive of it.
- **Plan storage:** The planner places successful plans in memory, indexed by the goals that they satisfy and the problems that they avoid.

The flow of control through these processes is simple. Goals are handed to the problem anticipator, which tries to predict any problems that might occur as a result of planning for them. If a problem is predicted, a goal to avoid it is added to the set of goals to be planned for. Once this is done, the goals are handed to the plan retriever, which searches for a plan that satisfies as many of the planner's goals as possible, including any goals to avoid any predicted problems. In order to do this, the plan retriever makes use of a memory of plans indexed by the goals they satisfy and the problems they solve. Once a base line plan is found, the plan modifier alters the plan to satisfy any goals that it does not already deal with. The alteration of the plan is done using a set of modification rules that are indexed by the goal to be added and the type of plan being altered.

After the plan is built it is handed to a simulator, which runs the plan using a set of rules concerning the effects of each action in CHEF's domain under different circumstances. If the plan runs without fail, it is placed in memory, indexed by the goals that it satisfies. If there are failures, the plan is handed to the plan repair mechanism. This process builds a causal description of why a plan has failed and then uses that description to find the repair strategies that will alter the causal situation to fix the fault. After the plan has been repaired, it is placed in memory, indexed by the goals it satisfies and the problems it now avoids. Because a problem has to be anticipated before it can be planned for, however, the planner has to do more than just store plans by the fact they they solve particular problems. It also has to build up a knowledge base that can be used to infer problems on the basis of the features in the world that predict them. The planner must decide which features in the input are responsible for the problem and mark them as such. This credit assignment is done whenever a failure occurs and the features that are blamed for the failure are linked to the memory of the failure itself. These links are later used to predict failures so that plans can be found that avoid them.

These six processes make up the basic requirements for a case-based planner. Plan retrieval, modification and storage are essential to the basic planning loop that allows old plans to be modified in service of new goals. A plan repair process is need for those situations in which plans fail. A process that assigns blame to the features responsible for failures is required for the planner to be able to later anticipate problems. And problem anticipation is needed in order for the planner to avoid making mistakes that it has already encountered.

In the sections that follow, each of these modules will be discussed in turn. These sections will discuss two examples: one in which CHEF creates and then repairs a faulty plan for a stir fry dish including beef and broccoli and one in which CHEF uses the knowledge gained from the first example to create a plan for a stir fry dish including chicken and snow peas. Most of this paper will attend to the second example in which the knowledge learned by CHEF in dealing with the first example is actually used, so as to show the power of the processes of problem anticipation, plan retrieval and plan modification. In discussing the processes of failure repair, plan storage and credit assignment, however, the first example is discussed. This is because CHEF learns from the problems encountered in dealing with this example and these three modules make-up CHEF's repair and learning abilities.

IV PROBLEM ANTICIPATION

CHEF's initial input is a set of goals to include different tastes and ingredients in a type of dish. The first step that CHEF takes in dealing with them is to try to anticipate any problems that might arise in planning for them. CHEF wants to predict problems before they occur so it can find a plan in memory that already avoids them.

The planner anticipates problems on the basis of a memory of past failures that is linked to the features that participated in causing them. These links are used to pass markers from the features in an input to the memory of failures that those features predict. When the features that are related to a past problem are all present, the memory of the failure is activated and the planner is informed of the possibility of the failure reoccurring.

For example, one of the failures the planner has in memory relates to an attempt to make a stir-fry dish with beef and broccoli. In making the dish, the liquid produced when stir frying the beef ruined the texture of the broccoli while it was cooking. The failure is indexed in memory by the features of the goals that interacted to cause it: the goal to include meat, the goal to include a crisp vegetable and the goal to make a stir fry dish. When these features are present in an input the planner can predict that the failure will occur again. Once a problem is predicted, the planner can add a goal to avoid the problem to the set of goals that will be used to find a plan.

In planning for the goals to include SNOW PEAS and CHICKEN in a STIR FRY dish, the planner is is reminded of the past failure it encountered in building the beef and broccoli plan. This failure was due to the fact that CHEF tried to stir fry some beef and broccoli together in a past plan, allowing the liquid from the beef to make the broccoli soggy. The surface features of the goal to include meat and the goal to include a crisp vegetable are the same, so the planner predicts that it will make this mistake again if it does not attend to this problem.

```
Searching for plan that satisfies input goals -
 Include chicken in the dish.
 Include snow pea in the dish.
 Make a stir-fry dish.

Collecting and activating tests.

 Is the dish STYLE-STIR-FRY.
 Is the item a MEAT.
 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
  crisp vegetables soggy."
 Reminded of a failure in the
 BEEF-AND-BROCCOLI plan.
 Failure = 'The vegetable is now soggy'
```

Once a failure has been anticipated, CHEF builds a goal to avoid

it and adds this goal to the set of goals which will be used to search for a plan. This set of goals is then handed to the plan retriever.

V PLAN RETRIEVAL

The function of the plan retriever is to find the best plan to use in satisfying a set of goals. It seeks what we call the "best match" in memory, the plan for a past situation that most closely resembles the current situation. In CHEF, this notion of "best match" is defined as finding a plan that satisfies or partially satisfies as many of the planner's most important goals as possible. Finding a raspberry soufflé recipe that can be turned into a strawberry soufflé or finding a beef stir fry dish that can be turned into one for pork. The plan retriever uses three kinds of knowledge in finding the best match for a set of goals: a plan memory that is indexed by the goals that the plans satisfy and the problems that they avoid, a similarity metric that allows it to notice partial matches between goals and a value hierarchy that allows it to judge the relative importance of the goals it is planning for.

When CHEF's retriever is searching for a past case on which to base its planning, it is searching for a plan that was built for a situation similar to the one it is currently in. The idea behind this search is that the solution to a past problem similar to the current one will be useful in solving the problem at hand. But this means that the vocabulary used to describe the similarity between the two situations has to capture the relevant aspects of the planning problems that the planner deal with. This vocabulary consists of two classes of objects: the goals in a situation, (which in the case of stored plans are satisfied by those plans) and the problems that have been anticipated (which in the case of the stored plans are avoided by them). CHEF gets its goals directly from the user in the form of a set of constraints that have to be satisfied. It gets information about the problems that it thinks it has to avoid while planning for those goals from its problem anticipator, which examines the goals and is reminded of problems that have resulted from interactions between similar goals in the past.

CHEF's domain is Szechwan cooking, so the goals that it plans for are all related to the taste, texture and style of the dish it is creating. The basic vocabulary that CHEF uses to describe the effects of its plans, and the effects that it wants a plan to accomplish include descriptions of the foods that it can include, (*e.g.,* Beef, chicken, snow peas and bean sprouts), the tastes that the user wants (*e.g.,* Hot, spicy, savory and fresh), the textures that the dish should include, (*e.g.,* Crunchy, chewy and firm) and the type of dish that the user is looking for, (*e.g.,* STIR-FRY, SOUFFLE, and PASTA). In searching for a past situation which might be useful, the plan retriever uses the goals that it is handed to index to possible plans. The plan it finds, then, will satisfy at least some of the goals it is looking for.

The problems that CHEF tries to avoid, by finding plans that work around past instances of them, also relate to the planner's goals. In searching for a plan, CHEF uses the predictions of any problems that it has anticipated to find plans that avoid those problems. Because plans are indexed by the problems that they solve, the planner is able to use these predictions to find the plans that will avoid the problems they describe. In searching for a base-line plan for the chicken and snow peas situation, CHEF searches for a plan that, among other things, avoids the predicted problem of soggy vegetables. Because the beef and broccoli plan is indexed by the fact that it solves the problem of soggy vegetables due to stir frying with meats, the plan retriever is able to find it on the basis of the prediction of this problem, even though the surface features of the two situations are dissimilar.

In searching for a plan that includes chicken and snow peas, the planner is also trying to find a plan that avoids the problem it has predicted of the vegetables getting soggy as a result of being cooked with the meat. The fact that it has predicted this problem allows it to find a plan in memory that avoids it, even though the plan deals with surface features that are not in the current input. The planner is able to find this plan even though a less appropriate plan with more surface features in common with the current situation, a recipe for chicken and green beans, is also in memory.

The planner's goals, including the goal to avoid the problem of the soggy vegetable are all used to drive down in a discrimination network that organizes past plans.

```
Driving down on: Make a stir-fry dish.
 Succeeded -
Driving down on:
  Avoid failure exemplified by the state
 'The broccoli is now soggy' in recipe BEEF-AND-BROCCOLI.
 Succeeded -
Driving down on: Include chicken in the dish.
 Failed - Trying more general goal.
  Driving down on: Include meat in the dish.
   Succeeded -
Driving down on: Include snow pea in the dish.
 Failed - Trying more general goal.
  Driving down on: Include vegetable in the dish.
   Succeeded -

Found recipe -> REC9 BEEF-AND-BROCCOLI
```

Here CHEF finds a past plan that avoids the problem due to goal interactions that has been predicted while still partially satisfying the other more surface level goals.

VI PLAN MODIFICATION

CHEF's memory of plans is augmented by the ability to modify plans for situations that only partially match the planner's present problems. CHEF's plan modifier, the module that handles these changes, makes use of a modification library that is indexed by the changes that have to be made and the plan that they have to be made in. The modification rules it has are not designed to be complete plans on their own, but are instead descriptions of the steps that have to be added and deleted from existing plans in order to make them satisfy new goals. Along with the modification rules are a set of *object critics* that look at a plan and, on the basis of the items or ingredients involved, correct difficulties that have been associated with those ingredients in the past.

The process used by CHEF's modifier is simple. For each goal that is not yet satisfied by the current plan, the modifier looks for the modification rule associated with the goal and the general plan type. If no modification rule exists for the particular goal, the modifier steps up in an abstraction hierarchy and finds the modification rule for the more general version of the goal. Once a rule is found, the steps it describes are added to the plan, merged with existing steps when possible.

If the goal in question is already partially satisfied by the plan, (this happens when the plan satisfies a goal that is similar to the current one), the planner does not have to go to its modification rules. It replaces the new item for the old item, removing any steps that were added by the old item's ingredient critics and adding any steps required by the new item's ingredient critics.

In altering the BEEF-AND-BROCCOLI plan to include chicken and snow peas, the planner has the information that both new ingredients are partial matches for existing ones and can be directly substituted for them. A critic under the concept CHICKEN then adds the step of boning the chicken before chopping.

```
Modifying new plan to satisfy:
     Include chicken in the dish.
Substituting chicken for beef in new plan.

Modifying new plan to satisfy:
     Include snow pea in the dish.
Substituting snow pea for broccoli in new plan.

Considering critic:
```

```
Before doing step: Chop the chicken
 do: Bone the chicken. - Critic applied.
```

VII PLAN REPAIR

Because CHEF cannot avoid making errors, it has to be able to repair faulty plans in response to failures. As it is, CHEF's plan repairer is one of the most complex parts of both the CHEF program and the case-based theory that it implements. It is complex because it makes the most use of the planner's specific knowledge about the physics of the CHEF domain and has to combine this knowledge with a more abstract understanding of how to react to planning problems in general.

Once it completes a plan, CHEF runs a simulation of it using a set of causal rules. This simulation is CHEF's equivalent of the real world. At the end of the simulation, it checks the final states that have been simulated against the goals of the plan it has run. If any goal is unsatisfied or if any state has resulted that CHEF wants to avoid in general, an announcement of the failure is handed to the the plan repairer.

CHEF deals with plan failure by building a causal explanation of why the failure has occurred. This explanation connects the surface features of the initial plan to the failure that has resulted. The planner's goals, the particular steps that it took and the changes that were made are all included in this explanation. This explanation is built by back chaining from the failure to the initial steps or states that caused it, using a set of causal rules the describe the results of actions in different circumstances.

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 [3], are similar in function to the critics found in HACKER [4] and NOAH [2]. Each TOP is indexed by the description of a particular type of planning problem and each organizes a set of strategies for deal 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 that is related to the *interactions* between plans and goals. This information cannot be tied to any individual goal or plan but is instead tied to problems that rise out of 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. 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, then, a planner actually finds the strategies that will help to fix that problems.

CHEF does not run into any failures while planning for the stir fry dish with chicken and snow peas. This is because it is able to avoid the problem of the liquid from the meat making the vegetables soggy by anticipating the failure and finding a plan the avoids it. It can only do this, however, because it has handled this problem already in that it built a similar plan that failed and was then repaired. This earlier plan is the same one that CHEF selected as the base-line plan for this situation, because it knows that this is a plan that it repaired in the past to avoid the same problem of liquid from the meat making the vegetables soggy that is now predicted as a problem in the current situation.

This past failure occurred when CHEF was planning for the goals of including beef and broccoli in a stir fry dish. CHEF originally built a simple plan in which the two main ingredients were stir fried together. Unfortunately, this plan results in the liquid produced by stir frying the beef making the vegetables soggy as they are cooking. This explanation of the failure in this example indexes to the TOP SIDE-EFFECT:DISABLED-CONDITION:CONCURRENT, a memory structure 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 twenty that the program knows about. All of these TOPs are associated with causal configurations that lead to failures and store strategies for fixing the situations that 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 has 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
```

These three strategies reflect the different changes that can be made to repair the plan. They suggest:

- ALTER-PLAN:SIDE-EFFECT: Replace the step that causes the violating condition with one that does not have the same side-effect but achieves the same goal.
- SPLIT-AND-REFORM: Split the step into two separate steps and run them independently.
- ADJUNCT-PLAN:REMOVE: Add a new step to be run along with a step that causes a side-effect that removes the side-effect as it is created.

In this case, only SPLIT-AND-REFORM can be implemented for this particular problem so the change it suggests is made. As a result the single stir fry step in the original plan in which the beef and broccoli were stir fried together is changed into a series of steps in which they are stir fried apart and joined back together in the final step of the plan.

Once a plan is repaired it can be described as a plan that now avoids the problem that has just be fixed. When it is stored in memory, then, it is stored as a plan that avoids this problem so it can be found if a similar problem is predicted.

VIII PLAN STORAGE

Plan storage is done using the same vocabulary of goals satisfied and problems avoided that plan retrieval uses. Once a plan has been built and run, it is stored it in memory, indexed by the goals it satisfies

and the problems it avoids. The plans are indexed by the goals that they satisfy so the planner can find them later on when it is asked to find a plan for a set of goals. They are also stored by the problems that they avoid so that CHEF, if it knows that a problem is going to result from some planning situation, can find a plan that avoids that problem.

The repaired BEEF-AND-BROCCOLI is indexed in memory under the goals that it satisfies as well as under the problems that it avoids. So it is indexed under the fact that is a plan for stir frying, for including beef and so on. It is also indexed by the fact that it avoids the problem of soggy vegetables that rises out the interaction between meat and crisp vegetables when stir fried. The fact that the plan is associated with the problem that it solves allows the plan retriever to later find the plan to use when confronted with the later task of finding a plan that avoids the problem of soggy vegetables that results when meats and crisp vegetables are stir fried together.

```
Indexing BEEF-AND-BROCCOLI under goals and problems:

If this plan is successful, the following should be true:
   The beef is now tender.
   The broccoli is now crisp.
   Include beef in the dish.
   Include broccoli in the dish.
   Make a stir-fry dish.

The plan avoids failure  exemplified by the
 state 'The broccoli is now soggy' in
 recipe BEEF-AND-BROCCOLI.
```

IX CREDIT ASSIGNMENT

CHEF's approach to failures is two fold. It repairs the plan to make it run and it repairs itself to make sure that it will not make the same mistake again. Part of assuring that the same mistake will not be repeated is storing the repaired plan so that it can be used again. But the fact that the original plan failed and had to be repaired in the first place indicates that CHEF's initial understanding of the planning situation was faulty in that it built a failure when it thought it was building a correct plan. When it encounters a failure, then, CHEF has to also find out why the failure occurred so that it can anticipate that failure when it encounters a similar situation in the future.

CHEF's makes use of the same causal explanation used to find its TOPs and repair strategies to figure out which features should be blamed for a failure. The purpose of this blame assignment is to track down the features in the current input that could be used to predict this failure in later inputs. This ability to predict planning failures before they occur allows the problem anticipator to warn the planner of a possible failure and allow it to search for a plan avoiding the predicted problem. The power of the problem anticipator, then, rests on the power of the process that figures out which features are to blame for a failure.

CHEF's steps through the explanation built by the plan repairer to the identify goals that interacted to cause the failure. After being pushed to the most general level of description that the current explanation can account for, these goals are turned into tests on the input. This allows the planner to later predict failures on the basis of surface features that are similar to the ones that participated in causing the current problem.

As a result of the beef and broccoli failure, a test on the texture of vegetables, is built and associated with the concept VEGETABLE because a goal for a crisp vegetable predicts this failure while goals for other vegetables do not. It is associated with VEGETABLE rather than BROCCOLI because the rule explaining the failure is valid for all crisp vegetables not just broccoli. Because any meat will put off the liquid like that which participated in the failure no test is needed and a link is built directly from the presence of the goal to the memory of the failure.

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

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

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

Goal to be activated = Avoid failure exemplified
by the state 'The broccoli is now soggy' in
recipe BEEF-AND-BROCCOLI.
```

These links between surface features and memories of failures are used later to predict the same problem when it is handed the goals to makes a stir fry dish with chicken and snow peas. This prediction is then used to find the plan that avoids the problem.

X CHEF

The idea behind CHEF is to build a planner that learns from its own experiences. The approach taken in CHEF is to use a representation of those experiences in the planning process. To make this possible, CHEF requires what any case-based planner requires, a memory of past events and a means to retrieve and store new them, a method for modifying old plans to satisfy new goals, a way to repair plans that fail, and a way to turn those failures into knowledge of how to better plan.

By choosing plans on the basis of the problems that they solve as well as the goals they satisfy, CHEF is able to avoid any problem that it is able to predict. By also treating planning failures as understanding failures and repairing its knowledge base as well as its plans, CHEF is able to predict problems that it has encountered before. And by using an extensive causal analysis as a means of diagnosing problems, CHEF is able to apply a wide variety of repairs to a single failure.

REFERENCES

[1] Fikes, R., and Nilsson, N., *STRIPS: A new approach to the application of theorem proving to problem solving,* Artificial Intelligence, 2 (1971).

[2] Sacerdoti, E., *A structure for plans and behavior,* Technical Report 109, SRI Artificial Intelligence Center, 1975.

[3] Schank, R., *Dynamic memory: A theory of learning in computers and people,* Cambridge University Press, 1982.

[4] Sussman, G., *Artificial Intelligence Series,* Volume 1: *A computer model of skill acquisition,* American Elsevier, New York, 1975.

[5] Wilensky, R., *META-PLANNING,* Technical Report M80 33, UCB College of Engineering, August 1980.