Explaining and Repairing Plans That Fail*

Kristian J. Hammond

Department of Computer Science,
The University of Chicago,
1100 East 58th Street, Chicago, IL 60637, USA

ABSTRACT

A persistent problem in machine planning is that of repairing plans that fail. One approach to this problem that has been shown to be quite powerful is based on the idea that detailed descriptions of the causes of a failure can be used to decide between the different repairs that can be applied.

This paper presents an approach to repair in which plan failures are described in terms of causal explanations of why they occurred. These domain-level explanations are used to access abstract repair strategies, which are then used to make specific changes to the faulty plans. The approach is demonstrated using examples from CHEF, a case-based planner that creates and debugs plans in the domain of Szechwan cooking.

While the approach discussed here is examined in terms of actual plan failures, this technique can also be used in the repair of plans that are discovered to be faulty prior to their actual running.

1. The Problem of Plan Failure

All planners face the problem of plans that fail. As a result, most planners have some mechanism for plan repair or debugging. These range from simple replanning and backtracking [5] to analytically driven critics [16] and meta-planning techniques [27]. The most powerful approaches to repair, however, have been those that make use of the notion that a planner should first explain and then debug any failures that are detected (e.g., [21, 29]). This work builds on a tradition initially established by Sussman [24] of approaching plan repair from the point of view of developing vocabularies to construct descriptions of planning problems that can be used to select specific and thus powerful repairs.

This paper presents an approach to plan repair that uses a causal description of *why* a failure has occurred to index to a set of strategies for repairing it. This approach makes use of a set of repair rules that are indexed by the causal descriptions of the problems that they solve. The repair rules themselves are empty frames that are instantiated using the specific steps and states of a problem at hand. This combination of abstract repair knowledge and specific state information results in a repair mechanism that is both general (in that it can be applied to a wide range of problems) and powerful (in that the instantiated repairs are aimed directly at the specific details of any given problem).

The approach is embedded in the computer program CHEF, a case-based planner that creates new plans in the domain of Szechwan cooking. The difference between the approach taken in CHEF and that taken by other planners stems from CHEF'S use of causal explanations of its own failures to focus on and select relevant repairs.

2. Repair in CHEF

CHEF'S approach to plan repair is based on its ability to explain its own failures. This process of failure repair can be broken down into five basic steps:

(1) Notice the failure.
(2) Build a causal explanation of why it happened.
(3) Use the explanation to find a collection of repair strategies.
(4) Instantiate each of the repair strategies, using the specific steps and states in the problem.
(5) Choose and implement the best of the instantiated repairs.

The reasons behind most of these steps are straightforward. CHEF has to notice a failure before it can react to it at all. It has to try each of its strategies in order to choose the best one. It has to implement

* This paper describes work done at the Yale Artificial Intelligence Lab under the direction of Hoger Schank. This work was supported by the Advanced Research Projects Agency of the Department of Defense and monitored by the Office of Naval Research under contracts N00014-82-K-0149. N00014-85-K-0108 and N00014-75-C-1111. NSF grant IST-8120451. and Air Force contract F49620-82-K-0010. It was also supported by the Defense Advanced Research Projects Agency, monitored by the Air Force Office of Scientific Research under contract F49620-88-C-0058, and the Office of Naval Research under contract N0014-85-K-010.

one of them to fix the plan. The only steps that might not seem as straightforward are the second and third steps of building an explanation and using it to find a structure in memory that corresponds to the problem and organizes solutions to it.

When CHEF encounters a failure in one of its own plans, it uses a set of causal rules to explain why the failure has occurred. This explanation includes a description of the steps and states that led to the failure as well as a description of the goals that were being planned for by those steps and states. This description is used to discriminate down to the set of abstract repair strategies appropriate to the general description of the problem. CHEF then tries to implement each of the different strategies and then choose the one best suited for the specific problem.

The problem that this technique addresses is one that crops up again and again in knowledge-intensive systems: the problem of the control of knowledge access (McDermott [14], Stefik [23] and Wilensky [27]). In this case, the problem takes the form of choosing which plan repair, among many possible repairs, should be applied to a faulty plan. The goal is to have a planner that is able to diagnose its own failures and then use this diagnosis to choose and apply the repair strategy that will result in a corrected plan. The approach described in this paper allows a planner to do this diagnosis and repair without having to do an exhaustive or even extensive search of the possible results of applying the different plan changes that the planner has available. Further, the specific repair is selected on the basis of its ability to correct an existing problem while avoiding the introduction of any new ones.

3. An Overview of CHEF

Before getting into the repair method used in CHEF, it is important to understand CHEF as planner[1]. CHEF is a case-based planner that, like other case-based reasoning systems, such as those of Carbonell [3] and Kolodner et al. [10. 11], builds new plans out of its memory of old ones. Its 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 basic approach is to find a past plan in memory that satisfies as many of the most important goals as possible and then modify that plan to satisfy the other goals as well.

Before searching for a plan to modify, CHEF examines the goals in its input and predicts any failures that might rise out of the interactions between the plans for satisfying them. This prediction is made on the basis of past problems that the planner has encountered rather than an exhaustive base of rules. When 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. In one CHEF example, the program predicts that stir frying chicken with snow peas will lead to soggy snow peas because the chicken will sweat liquid into the pan. In response to this prediction, the program searches its memory for a stir fry plan that avoids the problem of vegetables getting soggy when cooked with meats. In doing so, it finds a past plan for beef and broccoli that solves this problem by stir frying the vegetable and meat separately. The important similarity between the current situation and the one for which the past plan was built is that the same problem rises out of the interaction between the planner's goals, although the goals themselves are different. This similarity is what allows the planner to access the plan from the past, given the description of the goal interaction in the present.

The power of CHEF lies in its ability to anticipate and thus avoid failures it has encountered before. The topic of this paper, however, is its ability to repair these planning failures when it first encounters them. CHEF consists of six modules:

(1) An ANTICIPATOR predicts planning problems on the basis of the failures that have been caused by the interaction of goals similar to those in the current input.
(2) A *RETRIEVER* searches CHEF'S plan memory for a plan that satisfies as many of the current goals as possible while avoiding the problems that the ANTICIPATOR has predicted.
(3) A *MODIFIER* alters the plan found by the RETRIEVER to achieve any goals from the input that it does not satisfy.
(4) A *REPAIRER* is called if a plan fails. It builds up a causal explanation of the failure, repairs the plan and then hands it to the STORER for indexing.
(5) An *ASSIGNER* uses the causal explanation built by the REPAIRER to determine the features which will predict this failure in the future.
(6) A *STORER* places new plans in memory, indexed by the goals that they satisfy and the problems that they avoid.

[1] For a more detailed discussion of CHEF's planning mechanisms, see [8].

The important module for plan repair is the REPAIRER. This module is handed a plan only after it has been run in CHEF'S version of the real world, a "cook's world" simulation in which the effects of actions are computed. The rules used in this simulation are able to describe the results of the plan CHEF has created in sufficient detail for it to notice the difference between successful and unsuccessful plans. The final states of a simulation are placed on a table of results that CHEF compares against the goals that it believes that a plan should satisfy. Plans that fail in one way or another to satisfy all of their goals are handed to the REPAIRER for repair.

CHEF'S algorithm is an example of what McDermott has referred as *dependency-directed transformation* in which a preliminary plan is drawn from a library of plan schemas and then incrementally debugged, either in response to simulation or execution. Another example of this approach is Simmons' "Generate, Test and Debug" (GTD) [21] a problem solver that combines associational rules with a deep causal model. Given a problem, the associational rules are used to generate preliminary (and potentially buggy) solutions and then the causal model is used to debug them incrementally. The main difference between the two models lies in the attitude each takes towards its domain. One of Simmons' goals in GTD is a domain-independent model of planning, while the goal in CHEF was to develop a planner that learns (through the debugging of faulty plans) how better to act and react in its domain of interest. In some sense, the attitude taken in the development of CHEF was that there is a tradeoff between domain expertise and problem-solving generality, but that this expertise can be learned from weak methods.

4. Case-Based Reasoning

CHEF is part of a growing movement in Artificial Intelligence that involves the use of memory in planning (e.g., [1, 11]), problem solving (e.g., [15, 22]), language understanding [13] and diagnostic systems [12]. The main feature that links 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, PLEXUS [1] and MEDIATOR [11] have tried to address issues of planning from a dynamic memory. PLEXUS' memory was designed to provide nearly right plans for an execution-time improvisional 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 that occur 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 that rise out of classification errors which can be solved by building new categories in memory. CHEF, on the other hand, is aimed at problems of the interaction between planning steps that are 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, but limits it to those problems that 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 these approaches are needed in order to create a functioning case-based planner.

The major difference between CHEF and MEDIATOR is that of vocabulary. One of the major 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 was more closely linked to the features of the domains in which it operated. 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 between surface features and memories of existing problems that is used to anticipate those problems when those features are present.

Another line of research in case-based reasoning that has shown a great deal of promise has been in the area of legal reasoning [15]. 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 so as to find relevant cases in memory. This use of strategic modification has resulted in systems that can retrieve cases from memory that are similar to but strategically different from an initial input case. By finding those cases that differ along specific

dimensions, the system can reinforce its own arguments as well as anticipate problems that 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 [20].

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 work on using memory in parsing, Martin and Riesbeck [13] 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 through memory back down 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 [12] has proposed a case-based approach to diagnosis that 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 that allowed it to perform the same task using less computational power and 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.

5. Indexing of Repairs

5.1. TOPS in understanding and repair

CHEF'S approach to plan repair is based on the idea that the strategies for repairing a problem should be stored in terms of that problem. In much the same way that it makes sense to organize plans in terms of the goals that they satisfy so that they can be found when the goals arise, it makes sense to store plan repair strategies in terms of the situations to which they apply, so that they too can be found when those problems arise.

CHEF builds an explanation in response to every failure. These causal descriptions are used to discriminate between different structures in memory that describe the different interactions that can arise between steps in a plan. These structures are planning versions of the understanding TOPS suggested by Schank [17] to store information relating to complex interactions of goals. The idea behind these structures was simple: just as the concept underlying a primitive action such as ATRANS [19] could be used to store inferences relating to that action, structures relating to the interactions between these actions and the goals that they serve could also be used to store inferences that relate to the interaction. He also suggested that these structures could be used to organize episodes that they describe as well as the inferences that they relate to.

Schank argues that these structures are used for understanding in two ways: to supply expectations about a situation and to provide remindings of past situations that are similar to a current one that could be used in generalization. By having structures in memory that correspond to goal interactions rather than to individual goals or actions, an understander can have access to expectations that relate to those interactions and can learn more about them by generalizing across similar instances that they describe.

In CHEF, plan repairs are stored in terms of the problems that they solve using a set of planning TOPs that correspond to different planning problems. Each TOP organizes a set of repair strategies, which in turn suggest abstract alterations of the circumstances of the problem defined by the TOP. A TOP is defined not only by the series of steps that define the failure, it is also defined by the goals that those steps were originally designed to satisfy. Only those strategies that will fix the problem defined by a TOP, while at the same time maintaining the goals that are already satisfied by the existing steps, are found under that TOP. Any strategy that can be applied will repair the current problem without interfering with the goals that the steps involved with that problem still achieve.

> A planning TOP consists of the description of a planning problem and the set of repair strategies that can be applied to solve the problem.

Like Schank's TOPS, CHEF'S planning TOPS are memory structures that correspond to the interactions between the steps and states of a plan. Each planning TOP describes a planning problem in terms of a causal vocabulary of effects and preconditions. Instead of storing inferences about the situation described, however, each planning TOP stores possible repairs to the planning problem that it describes. Once a problem is recognized, the planner can use the repairs stored under the TOP that describes it to alter the faulty plan and thus fix the failure.

In CHEF, each TOP stores the repair strategies that will fix the faulty plans it describes. These

TOPS are not just descriptions of problems. They are descriptions of problems paired with the solutions that can be applied to fix the problems. The strategies under a TOP are those and only those alterations of the causal structure of the problem described by the TOP that can solve that problem. The strategies themselves are not specific repairs, they are the abstract descriptions of changes in the causality of the situation that CHEF knows about. Finding a TOP that corresponds to a problem means finding the possible repairs that can be used to fix that problem.

Each of CHEF'S TOPS is stored in memory, indexed by the features of the explanation that describes the problem with which the TOP deals. To get to the strategies that will deal with a problem. CHEF has to explain why it happened and then use this explanation to find the TOP and strategies that will fix the plan. This is a simple idea: the solution to any problem depends on the underlying causality of the problem. It makes sense, then, to index solutions to problems under the causal descriptions of the problem themselves so that these descriptions can be used to access the appropriate solutions.

Planning TOPs are memory structures that correspond to the interactions between the steps and states of a plan. Each planning TOP describes a planning problem in terms of a causal vocabulary of effects and preconditions. Instead of storing inferences about the situation described, however, each planning TOP stores possible repairs for the planning problem it corresponds to. Once a problem is recognized, the planner can use the repairs stored under the TOP that describes it to alter the faulty plan and thus fix the failure.

For example, the planning TOP, SIDE-EFFECT:BLOCKED-PRECONDITION describes situations in which the side effect of one step violates the preconditions for a later one. One case of this comes from Sussman's [24] example of a planner painting a ladder before painting a ceiling and then finding that the precondition for painting the ceiling, having an available ladder, has been violated by an earlier step in the plan that has left the ladder wet. This TOP is recognized because the planner can see that a step is actually blocked and can then build the explanation that the state that blocks it is the product of an earlier step and the state itself does not satisfy any goals. Once the planner has the TOP, it is then able to apply the different general strategies for repairing the faulty plan that are stored under the structure. In this case, the strategies are:

- ALTER-PLAN:PRECONDITION: Find a way to paint the ceiling that does not require a ladder, thus avoiding the problem of the failed precondition.
- ALTER-PLAN:SIDE-EFFECT: Find a way to paint the ladder that does not leave it wet, thus negating the blocking state.
- REORDER: Paint the ceiling before painting the ladder, thus running the step with the problematic precondition before it is blocked.
- RECOVER: Do something to dry the ladder, thus reestablishing the state before it is needed for a later step.

Each of these strategies is designed to break a single link in the causal chain that leads to a failure.

The planning TOPS discussed in this paper were developed for the CHEF program and are built out of a general vocabulary of causal interactions.[2] The problems that they describe include many of those discussed by both Sussman [24] and Sacerdoti [16], but the level of detail of TOPS allows a richer description than was possible using the vocabulary of critics suggested by either of them. As a result, a planner that uses TOPS to diagnose and repair planning problems is able to describe problems in greater detail and thus suggest a wider variety of repairs for each problem encountered.

5.2. Controlling repair

A TOP stores those, and only those, strategies that will solve the problem corresponding to the TOP without interfering with the goals that are satisfied by the steps included in the problem. As a result, CHEF is able to control its use of repairs so as to fix existing problems without interfering with the goals satisfied by other steps highlighted in the TOP. This is an important point that deserves an example.

In a failure encountered by CHEF while creating a strawberry souffle plan, the problem is diagnosed as a case of SIDE-EFFECT:DISABLED-CONDITION:BALANCE. This TOP is distinguished by the fact that the condition that is disabled is a balance condition and the state that disables it is a side effect. That the state that undermines the plan is a side effect is important in that it allows the strategies RECOVER and ALTER-PLAN:SIDE-EFFECT to be associated with the TOP. The first of these strategies suggests finding a step that will remove the side effect state before it interferes with a later step. The second strategy suggests finding a replacement for the step that caused the side effect with a step that

[2] For a detailed discussion of this vocabulary, see Section *9.*

accomplishes the goal of the original action without producing the undesired effect. Both of these strategies depend on the fact that the state in question is a side effect, that is, a state that does not satisfy any goals that the planner is trying to achieve.

If the situation were different (e.g., if the liquid produced by pulping the strawberries satisfied one of the planner's goals), these two strategies would no longer apply. Adding a step that would remove the added liquid would violate the goal that it achieved. Replacing the step that causes the liquid with one that does not would likewise violate the goal. Changing the goals changes the strategies that can be applied to the problem because some strategies will now interfere with those goals while fixing the problem. Such differences between situations are reflected in the TOPs that are found to deal with them. If the liquid served some goal but violated a balance condition, the TOP found would be DESIRED-EFFECT:DISABLED-CONDITION:BALANCE. Unlike the TOP SIDE-EFFECT:DISABLED-CONDITION:BALANCE, this structure does not suggest the strategies ALTER-PLAN:SIDE-EFFECT and RECOVER, because they would fix the initial problem only at the expense of other goals.

Another important point about the relationship between planning TOPS and the strategies they store is the problem of applicability. Every strategy that is stored under a TOP will, if it can be applied, repair the problem that originally accessed the TOP. There is no guarantee, however, that a strategy can be applied in every situation. In the case of a problem CHEF encounters while stir frying beef and broccoli together[3]; one strategy, ADJUNCT-PLAN, suggests finding a step that can be run concurrent with the STIR-FRY step that will absorb the liquid produced by the beef. Unfortunately, no such step exists in CHEF'S knowledge of steps. So ADJUNCT-PLAN, while it would have repaired the plan, cannot be implemented in this situation because the steps required to turn it into an actual change of plan do not exist. Each strategy describes a change that would fix a failed plan, but no strategy can be implemented if the steps that would make that change do not exist.

Each TOP describes a specific causal configuration that defines a problem. In turn, each of the strategies under a TOP describes one way to alter the configuration that will solve the problem described by the TOP. Each of the individual strategies suggests a change to one part of the overall configuration. Because the causal configurations that define these TOPS are built out of a vocabulary of interactions that is common to all of them, many share repair strategies. For example, one TOP corresponds to the situation in which the side effect of a step alters the conditions required for the running of a later step. Another corresponds to the problem of the side effect of a step itself being an undesired state. These two TOPS share the feature that a side effect of a step is interfering with the planner's goals, although in different ways. Because of this, they share the strategy ALTER-PLAN:SIDE-EFFECT which suggests finding a step to replace the original step in the plan that causes the side effect with one that achieves the same goals but does not cause the side effect. Each of these TOPS also organizes other strategies, but because some aspects of the problems that they describe are shared, they also share the strategy that suggests changes to that aspect of the problem.

> The repair strategies stored under a TOP are those, and only those that will repair the problem described by the TOP. Because each strategy alters one aspect of the problem, it may be associated with many TOPS that share that aspect.

Each of CHEF's TOPS has two components: the problem features used to index to it and the strategies that it suggests to deal with that problem. CHEF stores its TOPS in a discrimination network, indexed by the features that describe the problem that they correspond to. In searching for a TOP, CHEF extracts these same features from its explanation of the current problem, and then the TOP suggests the strategies it stores to solve that problem.

CHEF uses sixteen TOPs to organize its repairs. The details of each are discussed in Section 7. For now, it is only important to understand that each TOP is defined and indexed by the features of a particular planning problem and organizes the two to six strategies that can be applied to the problem it corresponds to. While there are clearly more TOPS that can be used in plan repair than CHEF knows about, those it has describe the planning problems that it is forced to deal with. An expanded domain with a different set of problems, and possible reactions to them, would require an equally expanded set of TOPS and strategies.

6. CHEF'S Repair Algorithm

As we mentioned in Section 2, CHEF's process of failure repair has five basic phases:

(1) Notice the failure.

[3] The liquid produced by stir-frying the beef causes the broccoli to become soggy.

(2) Build a causal explanation of why it happened.
(3) Use the explanation to find a collection of repair strategies.
(4) Instantiate each of the repair strategies, using the specific steps and states in problem.
(5) Choose and implement the best of the instantiated repairs.

In the sections that follow, we will discuss each of these phases as well as clarify each one with an example from the running of CHEF.

6.1. Noticing the failure

The first step in dealing with a failure is noticing it. CHEF is only able to recognize what we call *domain failures.* These are failures that can be characterized in terms of the operators and states in a domain. It is unable to recognize failures such as described by Wilensky's *metagoals* [27] (e.g., avoid wasting resources and avoid overly complex plans). This, however, is not because of any problem in terms of the theory of repair expressed in CHEF. It is just a byproduct of the emphasis of the program. CHEF is able to recognize three basic types of failures:

(1) failures of a plan to be completed because of a precondition failure on some step;
(2) failures of a plan to satisfy one of the goals that it was designed to achieve;
(3) failures of a plan because of objectionable results in the outcome.

In terms of its domain, CHEF can notice that a plan has stopped when the shrimp cannot be shelled because it is too slippery, that a plan to make a souffle fails because the batter has not risen and that a stir fried dish with fish fails because the fish has developed an iodine taste.

After CHEF has built a plan, it runs a simulation of it. This simulation is the program's equivalent of the real world rather than a part of the planning model, and a plan that makes it to simulation is considered complete. The result of this simulation is a table of descriptions that characterize the states of the ingredients used in the plan. Any compound objects created out of those ingredients are also described. The representation used by both CHEF and its simulator is a frame-like notation. In a current reimplementation of CHEF, we are doing the same work using a typed predicate calculus notation.

After running a plan to make a beef and broccoli dish, the table of results includes descriptions of the taste, texture and size of the different ingredients as well as the tastes included in the dish as a whole (Fig. 1).

Once a simulation is over, CHEF checks the states on this table against the goals that it believes should be satisfied by the plan that it has just run. These goals take the form of state descriptions of the ingredients, the overall dish and the compound items that are built along the way. No matter what its object, each goal defines a particular TASTE or TEXTURE for that object. Goals have the same form as the states placed on the simulator's result table, allowing CHEF to test for their presence after a simulation. CHEF tests for the satisfaction of goals by comparing expected states against those on the table of results.

Most of the failures that CHEF is able to recognize are related to the final result of the plan rather than the steps that the plan goes through on the way to that result. This is because CHEF notices failures by looking at that final product rather than monitoring the plan as it is being executed. The only failures that CHEF recognizes that are related to the running of the plan rather than its effects are those that result from the inability to perform a step because the conditions required for that performance are not satisfied.

```
(SIZE OBJECT (BEEF2) SIZE (CHUNK))
(SIZE OBJECT (BRGCCOLI1) SIZE (CHUNK))
(TEXTURE OBJECT (BEEF2) TEXTURE (TENDER))
(TEXTURE OBJECT (BROCCOLI1) TEXTURE (SOGGY))
(TASTE OBJECT (BEEF2) TASTE (SAVORY INTENSITY (9.)))
(TASTE OBJECT (BROCCDLI1) TASTE (SAVORY INTENSITY (5.)))
(TASTE OBJECT (GARLIC1} TASTE (GARLICY INTEHSITY (9.)))
(TASTE OBJECT (DISH) TASTE (AND (SALTY INTENSITY (9.))
                                (GARLICY INTENSITY (9.))
                                (SAVORY INTENSITY (9.))
                                (SAVORY INTENSITY (5.))))
```

Fig. 1. Section of result table for BEEF-WITH-BROCCOLI.

The easiest failure for CHEF to notice is the failure of a plan to finish because of a blocked precondition on a particular step. When this occurs, the simulator stops execution of the plan and informs CHEF that the plan has failed because of a blocked precondition. This is what happens in a plan for SHRIMP-STIR-FRY that CHEF created out of its memory of a FISH-STIR-FRY plan (Fig. 2). The problem with this dish is simply that a MARINATE step has been placed before a SHELL step, making the shrimp

too slippery to handle.

> One type of failure CHEF recognizes is the failure of a plan to run to completion because the preconditions for a step are blocked.

More often than not[4], CHEF'S plans run to completion. Even though a plan may run, this is no guarantee that it will run well or that it will succeed in doing all that it is supposed to do. After a plan finishes, CHEF must check its outcome to be sure that it has not failed to achieve the desired results.

CHEF evaluates its results along two dimensions. First, it has to check for any of the plan's goals that might not have been achieved. Second, it has to check for any generally objectionable states that might also have resulted from the plan.

Each of CHEF'S plans has a set of goals associated with it. These goals are generated by CHEF using the general role information attached to the general plan type and the specific ingredients of the plan itself. Each role specifies the expected contribution of its fillers and each ingredient provides the particulars of that contribution.

These goals take the form of state descriptions of the ingredients, the overall dish and the compound items that are built along the way. No matter what its object, each goal defines a particular TASTE or TEXTURE for that object. These goals all have the same form as the states placed on the simulator's result table, allowing CHEF to test for their presence after a simulation.

Once a plan has been run, the goals associated with the plan are searched for on the table of results built up by the simulator. If a goal is not present, then the plan has not succeeded in achieving it. This is

```
Simulating -> Marinate the shrimp in the soy sauce, sesame oil, egg white, sugar, corn
       starch and rice sine.

Unable to simulate -> Shell the shrimp.
       : failed precondition.

RULE: "A thing has to be handleable in order to shell it."

Some precondition of step: Shell the shrimp.
       has failed.
```

Fig. 2. A SHELL step blocked by unhandleable shrimp.

the second kind of failure that CHEF can recognize. It is the failure of a plan to achieve one of its goals. The failure in the beef and broccoli recipe is one of this type. CHEF finds the goal to have crisp broccoli associated with the plan, checks it against the states on the simulator's table of results and finds that the desired state is not there. Because the plan has failed to achieve one of its own goals, it is considered a failure (Fig. 3).

> The second type of failure that CHEF recognizes is the failure of a plan to achieve a goal it was designed to satisfy.

The final type of failure that CHEF can recognize is a failure that results from a state that has been included rather than excluded from the results of a plan. In this case, the state is one that the planner knows to be objectionable. Its inclusion in the results of a plan is a liability rather than an asset. In this situation, the plan may actually achieve all of the goals it is designed for but also results in states that are *a priori* objectionable.

To recognize these failures, CHEF uses descriptions of the particular states that it wants to avoid. Like positive goals, these states are of the same form as the states on the simulator's table of results so they can be tested for directly.

One example of this type of failure occurs as a result of CHEF'S first attempt to build a stir fried fish dish. The plan CHEF creates achieves all of the goals that it was designed for but also results in a dish that has the taste of iodine. This taste is a byproduct of the fish which was treated in the same way as the chicken that was originally in the dish. CHEF recognizes this by checking the states resulting from the plan against its knowledge of states to be avoided in general (Fig. 4).

> The last type of failure CHEF can recognize is the occurrence of an objectionable state in the result of a plan.

```
Checking goals of recipe -> BEEF-AND-BROCCOLI
```

[4] More often than not – (пер.) Очень часто, почти всегда.

```
Checking goal ->
       It should be the case that:  The broccoli is now crisp.

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

Recipe  ->  BEEF-AND-BRQCCOLI has  failed goals.
```

Fig. 3. Finding fault with BEEF-WITH-BROCCOLI.

```
Checking goals  of  recipe  ->  FISH-STIR-FRIED

Checking for negative features ->

Unfortunately:   The dish now  tastes   a  bad  taste.
In that:   The dish now tastes fresh,   like iodine,   like sea-food,
       like garlic,  hot,   oniony,   sweet,   nutty and salty.

Recipe -> FISH-STIR-FRIED has failed goals.
```

Fig. 4. Finding fault with FISH-STIR-FRIED.

6.2. Explaining the failure

Once a failure is noticed, it has to be explained. This is because the explanation of a plan failure is used to access the TOP and strategies that can be applied. The best way to organize plan repairs is under the descriptions of the problems that they solve so that the problem itself is a pointer to the solution. The best description in this case is a causal explanation of the problem. So the next step in dealing with a failed plan is to causally explain why the failure occurred.

This idea of explaining failure is not a new one. This approach to failure was suggested by Wilensky in his use of MAKE-ATTRIBUTION in PANDORA [28]. The goal of explanation in CHEF, however, is not simply to explain why the failure occurred. The goal is also to explain the presence of the steps and states that gave rise to the failure in terms of the planner's current goals. It is this combination of information that allows CHEF to select the set of strategies that will repair the plan while preserving the goals that it already satisfies.

> CHEF'S explanation of a failure is a causal description of the steps and states that led to it.

In the case of a failure due to a precondition that was not satisfied, the explanation is of the events that led up to the state that violated that condition. In the case of a failure due to an unsatisfied goal, it is an explanation of why the steps that normally lead to the goal state did not do so in this plan. In the case of a failure due to a result that includes an objectionable state, it is an explanation of why that state has come about. In all three cases, the actual format of the explanation is a dependency tree in which the planner's actions and the initial state of the world support the existence of the problematic states.

In general, the explanation of a failure is aimed at finding out:

- *What state constitutes the failure?*
- *What caused the failure?*
- *What goals were the steps involved in the failure aimed at achieving?*

By finding out the answers to these general questions, the planner can then find out how to respond to the failure while protecting the rest of the plan. This is the information that any planner would need in order to make an intelligent decision about how to react to a planning failure.

To build its explanations, CHEF uses the trace left by the forward chaining of the simulator. The links built by the simulator are a very simple set of causal connections that reflect the requirements of the domain. Steps are connected to the states that follow from them by RESULT links. States lead into new steps by filling slots and by satisfying PRECONDITIONS. The tests on preconditions are limited to tests on the texture, taste and existence of ingredients as well as their amounts and the relationship of those amounts to other ingredients. Failures are traced back from the failed states themselves through the steps that caused them, back to the conditions that caused the steps to fail, and so on back to the step that caused the unexpected condition itself.

> CHEF constructs its causal explanation of failures by chaining backward through the causal links built during the simulation of the failed plans.

It is important to note that we make no claims about the details of any process model involved with constructing these explanations. We are not claiming that we have solved or even made great advances in automatic diagnosis. We do, however, stand by our claim that the form of the diagnosis must be a causal description of the states and steps that actually led to a failure. We have done some work in the use of case-based techniques in the construction of explanations in [7, 9], but it is only tentative and discussion of it is beyond the scope of this paper.

CHEF'S movement through the causal network built up by the simulator is controlled by a set of explanation questions similar to those suggested by Schank [18]. These questions tell CHEF when to chain back for causes and when to chain forward for goals that might be satisfied by a particular state or step. Each answer tends to be used as the starting point for the next part of the search. For example, the answer to the question of what step caused an undesired state is the starting point in searching for the answer to the question of what condition altered the expected outcome of that step. The "questions", however, only serve to focus the basic search mechanism to the appropriate branches of the dependency structure. In the absence of a pre-existing support structure, the questions would act to guide a forward and backward chaining inference engine.

CHEF uses different sets of questions in explaining different types of failure. For failures in the results of a plan, CHEF has seven questions aimed at identifying the steps and states that have combined to cause an objectionable state or to block a desired one. For situations involving plans that cannot be completed because some precondition of a step is blocked, CHEF asks a somewhat different set of questions that are designed to identify the cause of the precondition violation. Both sets of questions, however, have the overall aim of diagnosing the problem so that the appropriate repair strategies can be found and applied.

> CHEF'S search through the simulator's causal trace is guided by a set of explanation questions that focus on the relevant steps and states. The process used to answer these questions is simply forward and backward chaining.

When a plan fails because of a faulty result, CHEF first asks the questions that will determine the steps that caused the failure. Then it asks the questions that will determine how those steps relate to the goals of the plan. The first set takes the form of backward chaining rules while the second takes the form of forward chaining rules.

The questions are:

- *What is the failure?*
  This identifies the particular problem state. In situations where the failure is an unsatisfied goal, the negation of the goal state serves as the answer. In situations where an objectionable state has come about, the state itself serves.
- *What is the preferred state?*
  Where a goal has not been achieved, the goal itself is the answer. Where an objectionable state has come about, the desired state of the object affected is focused on instead.
- *What was the plan to achieve the preferred state?*
  This question identifies the step that has actually caused the failure. The answer is found by chaining back from the failure to the step that actually caused it to come into being. To do this CHEF traverses a RESULT link back to the step.
- *What were the conditions that led to the failure?*
  With this question, CHEF is trying to identify the nonnormative conditions that caused the step to go wrong. Using the step that caused the failure as a starting point, CHEF searches back along PRECONDITION links to any states that had to be true for the original inference of the failure to be made.
- *What caused the conditions that led to the failure?*
  This question identifies the actual cause of the nonnormative conditions. Continuing to chain back. CHEF identifies the causes of the state that altered the predicted conditions and thus allowed the failure to result.
- *Do the conditions that caused the failure satisfy any goals?*
  This question directs CHEF'S attention forward. It looks to the effects of the condition that led to the failure for any goals that it directly or indirectly satisfies.
- *What goals does the step which caused the condition enabling the failure satisfy?*
  This question is similar to the last one in that it directs CHEF'S attention to the goals satisfied by the step that eventually led to the failure.

In explaining inability to complete a plan because of the failed preconditions of a step, CHEF uses

a slightly different set of questions.

- *What was the step that was blocked?*
  CHEF begins by identifying the blocked step itself.
- *What was the condition that disabled the step?*
  Checking the preconditions on the step, CHEF chains back to find the one that has failed.
- *What caused the precondition to fail?*
  Next it chains back along RESULT links to the step that caused the state which in turn violated the precondition.
- *What goals does the step which caused the condition disabling the step satisfy?*
  This question directs CHEF to chain forward to the goals satisfied by the step which caused the violating state.
- *Do the conditions that caused the failure satisfy any goals?*
  Just as the last question looked forward to the goals satisfied by the step this question looks forward to the goals satisfied by the violating state itself.
- *What goals does the blocked step serve?*
  With this question, CHEF again chains forward to find the goals served by the step that was blocked.

The answers to these questions give CHEF the same flexibility and assurance in this situation that it gets from the explanation of failures rising out of the results of a plan. It has the extended causal chain that leads back from the blocked step to the state that blocked it and then back to the step that caused that state. This gives it the knowledge of where it can change the plan. It also has the understanding of what goals the different steps satisfy. This gives it the knowledge it needs to alter a step or state while maintaining the goals that it serves.

> While CHEF asks different explanation questions in different situations, the questions it asks are always aimed at building a causal chain of the states and steps that led to the failure.

One good example of the kind of explanation that is built by CHEF is found in the case of the fallen strawberry souffle. The problem is that the added liquid from pulped strawberries disturbs the balance between the amount of liquid in the batter and the amount of whipped egg and flour used as leavening. Because there is too much liquid, the souffle falls. One important aspect of this situation is that the condition that prevents the souffle from rising is not just that there is too much liquid. It is that there is no longer a balance between the amount of liquid and the amount of leavening. When CHEF has to chain back to find the cause of this condition it has to check the two parts of the relationship against the normative case. The part that is irregular (in this case because it is new to the plan) is then traced down as the cause of the imbalance (Fig. 5).

This explanation gives CHEF the descriptors that will index to the planning TOP and repair strategies that will propose the appropriate set of solutions.

The fact that the PULP step has the side effect of producing liquid makes it a candidate for change. The fact that it enables the addition of the strawberries to the batter, thus making the batter taste like berries, means that any change that is made to this step to avoid the side effect has to include some addition that satisfies that goal. Likewise, the fact that the liquid is itself a side effect that satisfies no goals means that the planner can add steps that remove that side effect without having to worry about any goals that the condition served. The TOP and strategies that are found using these features reflect the constraints that they place on the type of repair that can be made to the plan.

> CHEF responds to failures by constructing causal explanations of why they have occurred.

This is a simple idea: the solution to a problem is based on the nature of the problem. Because of this, it makes sense to index different sets of solutions by the descriptions of different sorts of problems and then use the descriptions of the problems to find the solutions that are appropriate.

6.3. Getting the TOP

CHEF'S repair strategies are all stored under planning TOP'S, structures that correspond to different planning problems. The TOPS themselves are stored in a discrimination network, indexed by the features of the explanations they correspond to. The strategies organized under a TOP describe the alterations to the failed plans described by the TOP. These alterations are designed to repair the failure without interfering with the other goals in any plan that the TOP describes. The TOPS include structures such as SIDE-EFFECT:DISABLED-CONDITION:BALANCE and SIDE-FEATURE:ENABLES-BAD-CONDITION. The

strategies include changes such as REORDER the steps, REMOVE the condition, and SPLIT-AND-REFORM the step.

To get to a TOP, CHEF uses the structure and content of the dependency tree it has built to index through a discrimination network that organizes its TOPS. The features that are important in this discrimination include the nature of the violated condition, the temporal relationship between the steps, and the nature of the failure itself, as well as the relationship between the problematic state and the planner's goals.

ASKING THE QUESTION: 'What is the failure?'
ANSWER -> The failure is: It is not the case that: The batter is now risen.

ASKING THE QUESTION: 'What is the preferred state?'
ANSWER -> The preferred state is: The batter is now risen.

ASKING THE QUESTION: 'What was the plan to achieve the preferred state?'
ANSWER -> The plan was: Bake the batter for twenty-five minutes.

ASKING THE QUESTION: 'What were the conditions that led to the failure?'
ANSWER -> The condition was: There is an imbalance between the whipped stuff and the thin liquid.

ASKING THE QUESTION: 'What caused the conditions that led to the failure?'
ANSWER -> There is thin liquid in the bowl from the strawberry equaling 2.4 teaspoons, was caused by: Pulp the strawberry.

ASKING THE QUESTION: 'Do the conditions that caused the failure satisfy any goals?'
ANSWER -> The condition: There is thin liquid in the bowl from the strawberry equaling 2.4 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: Pulp the strawberry, establishes the preconditions for: Mix the strawberry with the vanilla, egg white, egg yolk, milk, sugar, salt, flour and butter. This in turn leads to the satisfaction of the goals: The dish now tastes like berries.

Fig. 5. Explanation in case of strawberry souffle failure.

The explanation of a failure is used to index through a discrimination net to a TOP that describes it in more general terms.

In the case of the strawberry souffle failure, the condition violated in the plan was a balance requirement between two amounts, and the condition that caused the imbalance was a side effect rather than a goal state. These two facts were very important in discriminating down to the TOP that corresponds to the situation. If the condition that caused the imbalance had not been a side effect or the requirement had not been one for a balance condition, a different TOP with different strategies would have been found (Fig. 6).

The TOP that describes the problem of the fallen souffle is indexed so that it can be found using the explanation of any situation in which a side effect of one step violates the conditions of a later step by causing an imbalance in a required relationship. Each of the strategies that is stored under SIDE-EFFECT:DISABLED-CONDITION:BALANCE is designed to deal with one aspect of this problem. If the problem had been different, a different TOP would have been found and a different set of strategies would have been suggested. The strategies under a TOP are limited by the types of changes CHEF is able to make and by the kinds of actions the domain allows. For instance, CHEF does not have the ability to

---

Searching for top using following indices:

Failure = It is not the case that: The batter is now risen,
Initial plan = Bake the batter for twenty-five minutes.
Condition enabling failure = There is an imbalance between the whipped stuff and the thin liquid.
Cause of condition = Pulp the strawberry.
The goals enabled by the condition = NIL
The goals that the step causing the condition enables = The dish now tastes like berries.

Found TOP TOP3 -> SIDE-EFFECT:DISABLED-CONDITION:BALANCE
TOP -> SIDE-EFFECT:DISABLED-CONDITION:BALANCE has 5 strategies associated with it:
ALTER-PLAN:SIDE-EFFECT

ALTER-PLAN:PRECONDITION
ADJUNCT-PLAN
RECOVER
ADJUST-BALANCE:UP

Fig. 6. TOP for strawberry souffle failure.

have other actors perform tasks for it, so the strategy of having a step performed by a different actor is not available to it.

> Each TOP organizes a set of strategies that, if implemented, will repair any problem described by the TOP.

Each of the strategies indexed under SIDE-EFFECT:DISABLED-CONDITION:BALANCE is also under other TOPS that share features with it. Each strategy suggests an alteration to the initial plan that will cause a break in a particular part of the causal chain that leads to the failure. Each change suggested by a strategy is, in principle, sufficient to repair the plan. So they are used individually and are not designed to be used in concert. Each strategy describes a single change to some link in the causal chain that leads to the failure. The strategies indexed under SIDE-EFFECT:DISABLED-CONDITION:BALANCE are:

- ALTER-PLAN:SIDE-EFFECT: Replace the step that causes the side effect with one that does not. The new step must satisfy the goals of the initial step.
- ALTER-PLAN:PRECONDITION: Replace the step that has the violated condition with a step that satisfies the same goals but does not have the same condition.
- ADJUNCT-PLAN: Add a new step that is run concurrent with the step that has the violated condition. This new step should allow the initial step to satisfy the goal even in the presence of the precondition violation.
- RECOVER: Add a new step between the step that causes the side effect and the step that it blocks. This step should remove the state that violates the precondition.
- ADJUST-BALANCE:UP: Adjust the imbalance between conditions by adding more of what the balance lacks.

Each of these five strategies suggests a change in the causal situation that will solve the current problem without affecting the other goals of the plan. They are those and only those changes that will alter the causal structure of the current faulty plan so as to remove that fault.

If the features of the problem had been different, the TOP and the strategies found would also be different. For example, if the condition violated by the side effect had not been a balance condition. CHEF would have found SIDE-EFFECT:DISABLED-CONDITTON, a simpler TOP that does not include the ADJUST-BALANCE:UP strategy. Likewise, if the condition satisfied goals of its own, DIRECT-EFFECT:DISABLED-CONDITION:BALANCE would have been selected, a TOP that lacks both ALTER-PLAN:SIDE-EFFECT and RECOVER because strategies that remove the condition would alter any plan to no longer satisfy the goal achieved by the condition. So, as the situation changes, the fixes that can be applied to it change, and thus the TOP and strategies that are found to deal with it change as well.

> Different problem descriptions allow access to different TOPs and different TOPS organize different strategies. The causality of a situation determines the strategies that will be suggested to deal with it.

Once it has found a planning TOP, CHEF has access to a set of strategies that, if implemented, will repair its current problem. It may not be the case, however, that every abstract repair will have a real implementation in every situation. It may also be the case that the change suggested by one strategy will be better than those suggested by others. So CHEF must test each strategy and decide which to apply to the particular problem. It does this by generating the changes to the plan that the TOP'S strategies suggest, choosing between those changes and then implementing the change it has chosen.

6.4. Applying the strategies

Once a TOP has been found, the strategies that are stored under it are applied to the problem at hand. For CHEF, applying a strategy means generating the specific change to the failed plan that is suggested when the abstract strategy is filled in with the specifics of the current situation. The idea here is to take an abstract strategy such as RECOVER and fill it in with the specific states and steps from the current problem. In this way a general strategy for repairing a plan becomes a specific change to the plan at hand.

To apply a strategy, CHEF fills the framework the strategy defines with the particulars of the current situation, turning it into a specific change rather than just an abstract alteration.

Each strategy has two parts: a test and a response. The test under each strategy determines whether or not the strategy has an actual implementation in the current situation. The response is the actual change that is suggested. In most cases, the result of the test run by the strategy is used in the response. For example, one of CHEF'S strategies is RECOVER, which suggests adding a new step between two existing ones that removes a side effect of the first before it interferes with the second. The test on RECOVER checks for the existence of a step that will work for the particular problem. The response is a set of instructions that will insert that step between the two existing ones. The action that is returned when CHEF searches for the step described by RECOVER is used in building the response (Fig. 7). The general format of the strategies is to build a test and then use the response to that test in building the set of instructions that CHEF has to follow in order to implement the change directed by the strategy.

These strategies fall into four general classes of changes: some alter the sequence of events in a plan, some break single steps into multiple ones, some add new steps and others replace old steps with new ones. Each of these classes of response has a slightly different type of test associated with it. Those involving

```
Asking question needed for evaluating strategy: RECOVER

ASKING ->
Is there a plan to recover from
       There is thin liquid in the bowl from the strawberry equaling 2.4 teaspoons.

There is a plan:  Drain the strawberry.

Response: Alter doing step: Pulp the strawberry.
          Do: Drain the strawberry.
```

Fig. 7. Test and response for RECOVER strategy.

changes to the order of steps in a plan only test for conflicts that the reordering might cause. Strategies that split steps test the step to check for any problems that splitting it might cause. Strategies that add steps check a library of actions for the existence of the steps that they describe. This library has actions indexed by their effects and preconditions which makes it possible for CHEF to search for a particular action that has the effect described by a strategy. Similarly, the strategies that direct the planner to replace steps check this library for actions that meet their precondition and effect specifications. These repair strategies make up the core of the strategies needed to deal with planning problems in general. They are not strategies that are related to the CHEF domain alone. Just as the planning TOPS are built out of a general vocabulary for describing different causal interactions, the repair strategies used by CHEF are general methods for altering plans. The vocabulary for describing TOPS is taken from a basic vocabulary of plan interactions and can describe a wide range of planning problems. Because the TOPS are products of the combination of these descriptors, however, each one is very specific. So the strategies stored under each can be equally specific. This means that the TOPS approach allows a planner to deal with a wide range of planning problems with very specific plan repairs.

This does not mean, however, that these strategies are in any way domain-specific. They are instead frames that are instantiated as specific changes to plans in response to problems that arise in all domains. With the general descriptive powers of the TOPS vocabulary, a wide range of problems can be described at this level of specificity.

Each strategy has a test that can be run to check for the conditions that have to hold for the strategy to be implemented. Each strategy also has a response, which gives CHEF the actual changes that it has to make to implement the strategy.

In building the tests and responses during the application of a strategy to a particular problem, CHEF uses the answers to its explanation questions to fill in the specific steps and states that the strategy will test and possibly alter. The tests and responses are actually empty frames that are filled with the specifics of the current explanation. The strategy RECOVER, for example, uses the answer to the question of what condition caused the current failure to construct its test and the answer to the question of what step caused that condition to build its response. This allows it to find a step that will remove the condition and run it immediately after the condition arises. The definition of each strategy refers to the answers to the explanation questions that are important to it, making it possible to build the specific test and response at

the appropriate time. Each definition begins with a binding of the existing explanation answers to variables that the strategy will then use to construct its query and response (Fig. 8). When the strategy is actually applied, the specific answers are inserted into the appropriate slots in the strategy structure.

> CHEF builds the changes suggested by a strategy by filling its framework in with the appropriate answers to its earlier explanation questions.

While each of the strategies that a TOP suggests for repairing a plan will result in a successful plan if implemented, there is no guarantee that an implementation will be found for a specific correction. In the beef and broccoli case, for example, one strategy that is suggested is ADJUNCT-PLAN. This directs the planner to find a step that can be run along with the original STIR-FRY step that will continuously remove the effects of stir frying the beef as they are created. While any such addition would repair the plan, because the liquid would be removed before it could affect the broccoli, CHEF does not have knowledge of such a step so the strategy cannot be implemented in this situation.

```
(del:strat recover
 bindings (*condition* expanswer-condition
           *step* expanswer-step)
 question (enter-text ("Is there a plan to recover from "
                       *condition*)
           test (search-step-memory *condition* nil)
           exit-text ("There is a plan" *answer*)
           fail-text ("No recover plan found"))
 response (text ("Response: After doing step: " *step* t
                 "          Do: " *answer*)
           action (after *step* *answer*)))
```

Fig. 8. Definition of RECOVER strategy.

While not all strategies will be useful in all situations, most of the time many changes are suggested that the planner can make. In the example of the strawberry souffle (Fig. 6). the five strategies associated with the TOP end up generating four possible changes that will repair the plan. CHEF generates all possible changes so that it can compare the specific changes and choose which one to actually implement on the basis of the changes themselves rather than on the basis of the abstract strategies (Fig, 9).

```
Applying TOP -> SIDE-EFFECT:DISABLED-CONDITION:BALANCE
to failure It  is not the case that:  The batter is now risen.
in recipe BAD-STRAWBERRY-SOUFFLE

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

ASKING ->
Is there an alternative to
       Pulp the strawberry.
that will enable
       The dish now tastes like berries.
which does not cause
       There is thin liquid in the bowl from the strawberry equaling 2.4 teaspoons

There is a plan: Use strawberry preserves.

Response: Instead of doing step: Pulp the strawberry
          do: Use strawberry preserves.

Asking questions needed for evaluating strategy: ALTER-PLAN:PRECONDITION

ASKING ->
Is there an alternative to
       Bake the batter for twenty-five minutes.
that will satisfy
       The batter is now risen.
which does not require
       It is not the case that: There is thin liquid in the bowl from the strawberry
       equaling 2.4 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 bowl from the strawberry equaling 2.4 teaspoons
that can be run with
       Bake the batter for twenty-five minutes.

There is a plan: Mix the flour with the egg, spices, strawberry, salt, milk, flour and
butter.

Response: Before doing step: Pour the egg yolk, egg white, vanilla, sugar, strawberry,
              salt, milk, flour and butter into a baking-dish
          Do: Mix the flour with the egg,  spices, strawberry, salt, milk, flour and
              butter.

Asking questions needed for evaluating strategy: RECOVER

ASKING ->
Is there a plan to recover from
       There is thin liquid in the bowl from the strawberry equaling 2.4 teaspoons

There is a plan: Drain the strawberry.

Response: After doing step: Pulp the strawberry
          do: Drain the strawberry.

Asking questions needed for evaluating strategy: ADJUST-BALANCE

ASKING ->
       Can we add more whipped stuff to BAD-STRAWBERRY-SQUFFLE

There is a plan: Increase the amount of egg white used.

Response: Increase the amount of egg white used.
```

Fig. 9. Strategies generating changes in strawberry soufflé example.

The changes suggested by the different strategies are aimed at altering different aspects of a problem. Some change the steps that caused a particular condition, some add steps that will remove the condition itself and others change the circumstances that make the condition a problem.

Each of CHEF'S strategies generates a change that is a combination of the abstract description of a repair provided by the strategy itself and the specifics of the failed plan. Because each TOP only stores those strategies that will repair the situation described by it and used to find it. any one of them will fix the plan if implemented. Because each TOP does have multiple strategies, however, not only must CHEF have a mechanism for generating these changes, it also must have one for choosing between them.

6.5. Choosing the repair

Once all of the possible repairs to a failed plan are generated. CHEF has to choose which one it is going to implement. To do this, CHEF has a set of rules concerning the relative merits of different changes. By comparing the changes suggested by the different strategies to one another using these heuristics, CHEF comes up with the one that it thinks is most desirable.

This set of heuristics is the compilation of general knowledge of planning combined with knowledge from the domain about what sort of changes will be least likely to have side effects. Some of these heuristics are closely tied to the domain, such as "It is easier to add a preparation step than a cooking step." and "It is better to add more of something that is already in the recipe than something new." Others are more domain-independent, such as "It is better to add a single step than to add many steps." and "It is better to replace a step than add a new step."

In the strawberry souffle example, the final repair that is chosen is to add more egg white to the recipe. This change is generated by the strategy ADJUST-BALANCE:UP which suggests altering the down side of a relationship between ingredients that has been placed out of balance. This repair is picked because it is the least violent change to the plan that can be made and has the least likelihood of creating one problem as it solves another.

CHEF chooses between the competing changes suggested by different strategies

using a set of repair heuristics designed to find the most reliable alteration.

6.6. Confirming the change

Once a change is selected. CHEF implements the change using its procedural knowledge of how to add new steps, split steps into pieces, remove steps and add or increase ingredients. These functions are the same as those used to implement the plan changes made by the MODIFIER when it adds new goals to existing plans.

In the strawberry souffle situation the final change is marginal, which is the best sort of change if it actually can repair the problem. The only difference between the original strawberry souffle plan and the new one is that the new one has more egg white in it. Figure 10 shows the change generated by the strategy ADJUST-BALANCE:UP.

> CHEF implements the changes suggested by a strategy using the same alteration functions used by the MODIFIER to add new goals to plans.

Once a change is made, the plan is simulated again to assure that the change has led to the expected result. If another failure occurs, it is once again passed through the repair process of explaining the failure, finding the TOP and applying the appropriate strategy. Although past changes will not be marked explicitly, the fact that they participate in satisfying a goal will stop the REPAIRER from removing them. For example, in order to fix a fish and peanuts recipe to avoid the problem of the iodine taste of the fish. CHEF adds the step of marinating it in rice wine. Unfortunately, this causes the fish to become soggy. In

```
Implementing plain -> Increase  the amount  of  egg white used.
Suggested by  strategy ADJUST-BALANCE:UP

New recipe  is -> STRAWBERRY-SOUFFLE

            STRAWBERRY-SOUFFLE

       Two teaspoons of vanilla A half  cup of flour
       A quarter cup of  sugar
       A quarter teaspoon of  salt
       A half cup of milk
       Two  cups  of milk
       One piece of  vanilla bean
       A quarter cup of butter
       Five  egg yolks
       Six egg whites
       One cup of  strawberry
```

Fig. 10. Change generated by ADJUST-BALANCE:UP strategy.

trying to fix the new failure, CHEF recognizes that it cannot just remove the marinate step, because it satisfies a goal of removing the iodine taste. Instead of removing the step CHEF replaces it with another step that satisfies the same goal, removing the iodine taste of fish, but does not add anymore liquid. Rather than marinating the fish in rice wine, it marinates it in crushed ginger.

If the plan is successful, it is then indexed in memory like any other plan, except it is marked by the fact that it avoids a particular kind of failure that can be used when that failure is predicted to occur at some later date.

Because CHEF uses its plans again and again, just building them is not enough. It also has to place them in memory so that they can be accessed again. While this paper does not stress learning, it is important to note that CHEF is designed to use the explanation generated while repairing a failure to discern which features will be predictive of problems of this type. This knowledge, combined with the existence of the new plan that repairs the problem, allows CHEF to *anticipate and avoid* the problem in later planning.

If a new plan is built out of one that avoids a failure, the fact that it too avoids the same failure is associated with it. When the new plan is stored in memory, it too is stored by the token which is related to the prediction of the failure. This means that any one failure can have many plans in memory that can be used to avoid it. Each one of these is indexed by the token related to the failure and is also indexed by the goals that it satisfies and any other failures that it also avoids. Every time a once failed plan is stored in memory, the token that is associated with the prediction of a failure is used to index it in memory. This closes the repair loop, allowing the prediction of a failure to provide the information needed to find the

plan that will avoid it.

7. CHEF'S TOPS

The repair algorithm is only part of the overall approach to repair used in CHEF. An equally if not more important aspect is the theory of the content of the types of problems that tend to be encountered and the different repairs that remove them. This section will outline the different problems that rise out of the interactions between the steps and goals of a plan (represented by planning TOPS). The sections that follow will also look at the various repairs that can be applied as well as the other uses to which TOPS can be put.

CHEF's TOPS fall into five categories or families of problems. Some have to do with side effects of a step interfering with later steps, some with the features of objects enabling bad effects, and some with the problems of blocked preconditions. The TOPs within each family share features with one another. The fact that they describe similar problem situations means they must share those strategies that are designed to repair the aspects of the problem that they have in common.

- SIDE-EFFECT, or SE, TOPS describe problems that are generated by the side effects of STEPS.
- DESIRED-EFFECT, or DE, TOPS describe problems that are generated by the desired effects of STEPS.
- SIDE-FEATURE, or SF, TOPS describe problems generated by the features of OBJECTS that do not satisfy any goals.
- DESIRED-FEATURE, or DF, TOPs describe problems generated by the goal-satisfying features of OBJECTS.
- STEP-PARAMETER, or SP, TOPs describe problems generated by improperly set STEP parameters.

CHEF'S TOPs fall into families of configurations whose members correspond to similar causal situations and organize similar repair strategies.

7.1. SIDE-EFFECT (SE)

The most commonly used TOPS in CHEF are from the family of TOPs that describe problems with the side effects of one step interfering with the overall plan in one way or another. There are five TOPS that belong to the general family of SIDE-EFFECT TOPS (SE):

- SE:DISABLED-CONDITION:CONCURRENT: The side effect of a step disables its own success requirements or the requirements of a concurrently run step.
- SE:DISABLED-CONDITION:SERIAL: The side effect of a step disables the success requirements of a later step.
- SE:DISABLED-CONDITION:BALANCE: The side effect of a step disables a success requirement of a later step by causing an imbalance in a required relationship.
- SE:BLOCKED-PRECONDITION: The side effects of a step violate the preconditions of a later step that are required for it to run at all.
- SE:GOAL-V1OLATION: The side effects of a step are in themselves an undesired state.

SE TOPs describe situations in which the side effects of a step interfere with the successful running of a plan.

7.1.1. SE:DTSABLED-CONDITION:CONCURRENT (SE:DC:C)

This TOP describes situations in which a side effect of a step interferes with the satisfaction conditions of the step itself or of another step that is being run concurrent with it. Here the side effect has to occur as the step is being run, not just at the end. Likewise, the condition that is violated is not just a precondition, it is a condition that has to be true for the entire duration of the step. In CHEF these are referred to as *developing side effects* and *maintenance conditions.* One example of this situation is in the case of the BEEF-AND-BROCCOLI plan in which the liquid generated by stir frying the beef violates the maintenance condition on the step to stir fry the broccoli that the pan remain dry. As a result, the plan fails because the broccoli becomes soggy.

The indices for SE:DC:C are:

- There is a failure.
- The failure is caused by a violated condition.
- The step that caused the state violating the condition is concurrent with the step which has resulted in the failure.
- The condition satisfies no goals.

- The step that causes the condition satisfies at least one goal directly or indirectly.

The strategies stored under SE:DC:C are:

SPLIT-AND-REFORM,
ALTER-PLAN:SIDE-EFFECT,
ADJUNCT-PLAN:REMOVE,
ADJUNCT-PLAN: PROTECT,
ALTER-PLAN:PRECONDITION.

Each of these strategies aims its attack at one part of the causal chain that led to the failure. For example, SPLIT-AND-REFORM suggests breaking the self-defeating single step into a series, ALTER-PLAN:SIDE-EFFECT suggests that a different step be used that does not produce the violating condition, and ADJUNCT-PLAN:REMOVE suggests that a new step be added that removes the effects altogether.

7.1.2. SE:DISABLED-CONDITION:SERIAL (SE:DC:S)

The complement to the TOP SE:DC:C is SE:DC:S, which differs from its sibling in that the step that causes the problem side effect occurs before the step with which the side effect interferes. The problem condition is still a side effect, but the steps involved are separate so there are more options than in the case of SE:DC:C.

The indices to this TOP differ from those of SE:DC:C only by the fact that the two steps involved are not run at the same time:

- There is a failure.
- It is caused by a violated condition.
- The step that would have achieved the desired state follows the step that caused the violating conditions.
- The condition satisfies no goals.
- The step that causes the condition satisfies at least one goal directly or indirectly.

The fact that two steps are separated in time gives the planner more options than if they had been at the same time. The strategies indexed under the TOP reflect this added flexibility:

REORDER,
ALTER-PLAN: SI DE-EFFECT,
ALTER-PLAN:PRECONDITION,
RECOVER,
ADJUNCT-PLAN:REMOVE,
ADJUNCT-PLAN: PROTECT.

7.1.3. SE:DISABLED-CONDITION:BALANCE (SE:DC:B)

A slightly different situation in which a side effect causes problems is defined by SE:DC:B. This is the situation in which the side effect of a step produces the effect of changing an existing state so as to increase an amount, in turn causing an imbalance in a relationship required for a later step. Unlike SE:DC:S, the condition caused by the earlier step is already accounted for by the plan, but not completely. The fallen souffle in the STRAWBERRY-SOUFFLE plan is an example of this problem. The added liquid from the pulped strawberries causes an imbalance between liquid and leavening in the batter which leads to a fallen souffle.

The difference between the features that access this TOP and those that access SE:DC:S is clear. SE:DC:B is found when the condition that is violated is a balance relationship that has to hold for a step to succeed.

- There is a failure.
- There is a violated balance condition on the step that caused it.
- The step that would have achieved the desired state follows the step that caused the violating conditions.
- The condition satisfies no goals,
- The step that causes the condition satisfies at least one goal directly or indirectly.

The strategies organized by the TOP are also the same, with the addition of two new strategies,

ADJUST-BALANCE:UP,
ADJUST-BALANCE:DOWN.

7.1.4. SE:BLOCKED-PRECONDITION (SE:BP)

SE:BP describes the situation in which the side effect of a step actually halts the running of the plan. This occurs in the case of the first stir fried shrimp dish CHEF creates in which the MARINADE step that is run before the SHELL step blocks the precondition on SHELL that the shrimp be handleable.

The features used to access this TOP are:

- There is a blocked precondition.
- The condition that caused the block was caused by an earlier step.
- The condition satisfies no goals.
- The step that causes the condition satisfies at least one goal directly or indirectly.

The four strategies associated with this TOP are:

RECOVER,
REORDER,
ALTER-PLAN PRECONDITION,
ALTER-PLAN:SIDE-EFFECT.

7.1.5. SE:GOAL-VIOLATION (SE:GV)

The TOP SE:GV describes the situation in which the side effect of a step itself is an undesired state in the present circumstances. The state generated by the step does not interfere with other steps, it is just objectionable in its own right.

The features used to index to this TOP are:

- There is a failure.
- The step that caused it has no violated conditions that the planner can identify.

The strategies stored under this TOP are:

RECOVER,
ALTER-PLAN:SIDE-EFFECT,
ADJUNCT-PLAN:REMOVE.

7.2. DESIRED-EFFECT (DE)

A somewhat different class of TOPS are those that describe situations in which the desired effect of a step, that is a state that either directly satisfies a goal or enables the satisfaction of a goal, causes problems later in the plan. The main difference between DESIRED-EFFECT and SIDE-EFFECT TOPS is that DE TOPS do not suggest strategies designed to remove or avoid the effects that are interfering with the plan. To do so would be to undercut the goals satisfied by the interfering state.

There are four DE TOPS. They are parallel to the first four SE TOPS that were discussed in the last section. But because these correspond to situations in which the interfering state satisfies a goal, the strategies that they store are aimed at altering the aspects of the problem that are not related to the desired effect itself. The DE TOPS are:

- DESIRED-EFFECT:DISABLED-CONDITION:CONCURRENT: A desired effect of a step disables its own success requirements.
- DESIRED-EFFECT:DISABLED-CONDITION:SERIAL: A desired effect of a step disables the success requirements of a later step.
- DESIRED-EFFECT:DISABLED-CONDITION:BALANCE: A desired effect of a step disables a success requirement of a later step by causing an imbalance in a required relationship.
- DESIRED-EFFECT:BLOCKED-PRECONDITION: The desired effects of a step violate the preconditions of a later step that are required for it to run at all.

CHEF does not use the TOP DESIRED-EFFECT:GOAL-VIOLATION because it defines a situation that it never has to encounter. DE:GV describes a situation in which the goal-satisfying effect of a step is itself a state the planner wants to avoid. While this can happen in some planning situations, it is never the case that CHEF has to deal with states that both meet and violate its goals.

> DE TOPS describe situations in which the desired effects of a step interfere with the successful running of a plan.

7.2.1. DE:DISABLED-CONDITION:CONCURRENT (DE:DC:Q)

This TOP describes the ultimate self-defeating plan. The desired effects of a step infer with the conditions that it itself requires to succeed. Because the violating state is also serving the planner's goals, strategies such as ADJUNCT-PLAN:REMOVE and ALTER-PLAN:SIDE-EFFECT cannot be applied because they would negate a state that the planner wants to maintain. Only those strategies that are aimed at correcting the situation without altering the part of the causal chain leading to the failure that includes the violating state are included under this TOP.

The features that are used to find this TOP are:

- There is a failure.
- There is a violated condition on the step that caused it.
- The plan to achieve the desired state itself caused the violating conditions.
- The condition satisfies at least one goal directly or indirectly.

The strategies stored under this TOP are:

ALTER-PLAN:PRECONDITION,
SPLIT-AND-REFORM,
ADJUNCT-PLAN: PROTECT.

7.2.2. DE:DISABLED-CONDITION:SERIAL (DE:DC:S)

Like its brother DE:DC:C, the TOP DE:DC:S is aimed at situations in which the desired effect of a step interferes with the plan rather than a side effect that can be removed or avoided altogether. This TOP describes the situation in which the desired effect of a step violates the satisfaction conditions of a later step.

The features that index to it are:

- There is a failure.
- There is a violated condition on the step that caused it.
- The step that would have achieved the desired state is different from the step that caused the violating conditions.
- The condition satisfies at least one goal directly or indirectly.

The fact that two steps are separate in time gives the planner more freedom than if they took place in parallel. The strategies under the TOP reflect this:

REORDER,
ALTER-PLAN:PRECONDITION,
RECOVER,
ADJUNCT-PLAN:PROTECT.

7.2.3. DE:D1SABLED-CONDITION:BALANCE (DE:DC:B)

A variant of the situation in which a relationship between two states is put out of balance is DE:DC:B, which describes the situation in which the culprit is the desired effect of a step rather than a side effect.

DE:DC:B is indexed by the following features:

- There is a failure.
- There is a violated balance condition on the step that caused it.
- The step that would have achieved the desired state is different from the step that caused the violating conditions.
- The condition satisfies at least one goal directly or indirectly.

The strategies organized by the TOP reflect the fact that the state causing the problems is a desired one by including only those that deal with the problem without interfering with that state. They are:

ALTER-PLAN PRECONDITION,
ADJUNCT-PLAN: PROTECT,
ADJUST-BALANCE: UP.

7.2.4. DE:BLOCKED-PRECONDITION (DE:BP)

The TOP DE:BP describes the situation of one of the classic problems in machine planning: the problem of ordering the steps required to build a three-block tower [24]. In this problem, the planner has the dual goals of having block *A* on block *B,* (ON *A B*), and block *B* on block *C* (ON *B* C). Unfortunately, if it starts the tower by putting *A* on *B,* (MOVE *A B),* before putting *B* on *C.* (MOVE *B* C), it is unable to continue because of the precondition on the MOVE step that the block being moved must have a clear top (Fig. 11). Because *A* is on *B*. *B* cannot be moved, but having *A* on *B* is also one of the planner's goals. The effects of the first step violate the preconditions for the later one. This is the situation described by DE:BP. The desired effect of one step blocks the preconditions required to perform another step, and both steps satisfy one or more of the planner's goals. The problem is how to change the situation so that both parts of the plan can be run.

The solution that has been suggested in the past [16, 24] is to repair the plan by reordering the steps, putting *B* on *C* before putting *A* on *B.* This solution is the same as one of the two suggested by the TOP DE:BP. But DE:BP also suggests the strategy of finding some way to move the *B* block that does not require that it has a clear top. At the very least the approach of splitting the process of plan repair into diagnostic and prescriptive stages has allowed a gain of one new solution to this classic problem. But, quite truthfully, this is not a particularly exciting gain.

A more exciting gain lies in the vocabulary used to describe this problem at all. The vocabulary suggested by Sussman in building HACKER describes this situation as one of *Brother-goal-clobbers-brother-goal,* While this description covers cases of DE:BP, it also covers cases of SE:BP where the state

blocking the goal is not a protected one. Because of this, HACKER would suggest the limited solution REORDER in cases described by SE:BP where CHEF would be able to suggest the strategies of RECOVER, ALTER-PLAN:SIDE-EFFECT, ALTER-PLAN: PRECONDITION, ADJUNCT-PLAN:REMOVE as well as REORDER. Because CHEF is able to distinguish between situations in which an interfering state serves some goals and those in which it does not, it is able to know when it can apply more powerful strategies than those planners that could not describe problems in these terms.

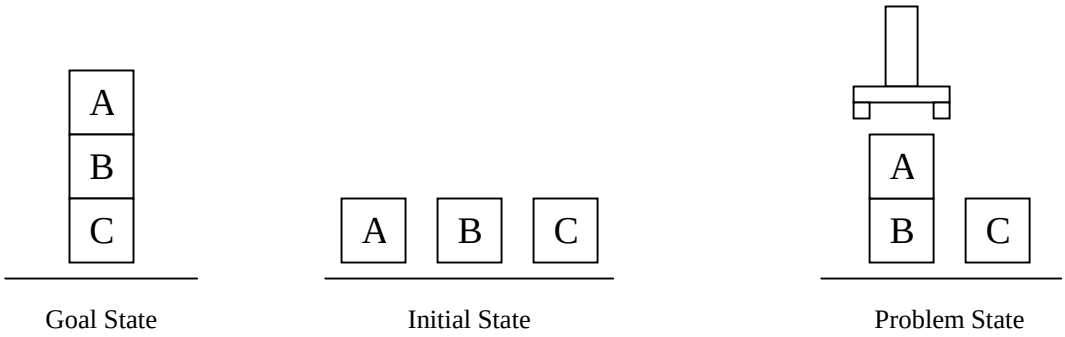

Fig. 11. The three blocks problem.

This is less a point about TOPS and more a point about the level of detail used to define them and describe different causal situations. Because the description of the situations leading to failures include distinctions between side effects and desired effects, CHEF is able to distinguish between cases where different strategies apply where earlier planners could not. Because it has knowledge of exactly what steps and states in the problem have to be protected, it is able to make alterations that other planners would not have the assurance to make.

The features that are used to index to DE:BP are almost identical to those used to access SE:BP. but the single difference between them has a great effect on the strategies that CHEF can apply to deal with the different situations. The features that access DE:BP are:

- There is a blocked precondition.
- The condition that caused the block was caused by an earlier step.
- The condition satisfies at least one goal directly or indirectly.

Because the planner has to protect the goals satisfied by the blocking condition it only has two strategies that it can apply to repair the situation:

REORDER,
ALTER-PLAN PRECONDITION.

7.3. SIDE-FEATURE (SF)

In many situations, a failure will rise out of a problem having to do with a particular feature of an item in a plan rather than having to do with the effects of one step relating to the effects of another. This is the type of situation described by TOPS in the SIDE-FEATURE family of structures. These SF TOPS relate to situations in which the features of new items that have been added to plans interfere with the goals that the plans are trying to achieve. This family of TOPS is called "SIDE-FEATURE" because the TOPS in it all describe situations where a feature of an object that does not satisfy a goal interferes with the plan. This is opposed to the DESIRED-FEATURE TOPS that describe situations in-which the desired feature of an object disrupts a plan.

There are three TOPS in this family that are actually used by CHEF:

- SIDE-FEATURE:ENABLES-BAD-CONDITION: An object's feature enables a step to have a faulty result.
- SIDE-FEATURE:BLOCKED-PRECONDITION: An object's feature disables a required precondition on a step.
- SIDE-FEATURE:GOAL-VIOLATION: An object's feature is itself an objectionable state.

SF TOPS describe situations in which a non-goal satisfying feature of an item interferes with the successful running of a plan.

7.3.1. SF:ENABLES-BAD-CONDITION (SF:EBC)

The TOP SF:EBC describes situations in which some feature of an object enables a step to have a bad result. The failure CHEF encounters in building FISH-STIR-FRY is an example of this sort of situation.

The taste of the fish interacts with the stir fry step to create an iodine taste. The failure is caused by a feature of the fish itself rather than by the side effect of a step leading into a later one.

The features of the situation that lead to recognizing this TOP are as follows:

- There is a failure.
- There is a violated condition on the step that caused it.
- The plan to achieve the desired state has a violated satisfaction condition.
- The violating state satisfies no goals.
- The condition is an inherent feature of an object.

Because there is only one step involved in this type of situation, the strategies to deal with the failure are centered around it. But because there is an identifiable state that is related to an object feature, there is also a strategy that is aimed at removing that feature:

ALTER-PLAN:PRECONDITION,
ADJUNCT-PLAN:PROTECT,
REMOVE-FEATURE.

7.3.2. SF:BLOCKED-PRECONDITION (SF:BP)

Sometimes a new object is added to a plan that the steps of the plan are not designed to deal with at all. If the MODIFIER allows this to happen, it is possible that certain steps will be blocked altogether by some feature of the new object. This is the situation described by SF:BP. One example of this is the problem of adding lobster to a fish dish, replacing it for the existing fish. In this situation, the planner has to deal with the problem of the shell, either when building the plan or later when a failure points to the problem. Because the texture of the lobster does not meet the preconditions for a CHOP step, the plan cannot be run without alteration. The problem is not that one step is interfering with another. It is that one of the items in the plan does not meet the requirements for running one of its steps.

The features that make it possible to recognize this situation are:

- There is a blocked precondition.
- The condition that caused the block is a feature of an object.
- The feature satisfies no goals.

The strategies that this TOP organizes are again aimed at changing the violated step itself and altering the feature of the object involved:

ALTER-FEATURE,
ALTER-PLAN:PRECONDITION.

7.3.3. SE:GOAL-VIOLATION (SF:GV)

Sometimes the problem with an item is not in that it leads to difficulties, but instead that it has a feature that by itself is a problem. This is the situation described by SF:GV. One example of this situation that CHEF deals with is the problem it encounters while trying to create duck dumplings. In this case, the fat from the duck itself is an objectionable feature that violates one of the planner's goals. This TOP has only those strategies that are aimed at removing the features from an item that are themselves objectionable and has no suggestions about changing the existing steps in the plan.

This situation is easy to recognize:

- There is a failure.
- The failure is an inherent feature of an object.

The strategies suggested in this situation are aimed only at changing the object itself:

ALTER-ITEM,
REMOVE-FEATURE.

7.4. DESIRED-FEATURE (DF)

Another family of TOPs related to the features of items rather than the interactions of steps is DESIRED-FEATURE (DF). DF TOPs all describe situations in which a feature of an object that actually satisfies a goal causes trouble. Unlike SIDE-FEATURE TOPs, DF TOPs store strategies that only alter the steps affected by the problem feature.

There are only two TOPs in this family that CHEF uses:

- DESIRED-FEATURE:DISABLED-CONDITION: A goal-satisfying feature of an object enables a faulty result from a step.
- DESIRED-FEATURE:BLOCKED-PRECONDITION: A goal-satisfying feature of an object actually blocks a required precondition of a step.

DF TOPs describe situations in which a goal-satisfying feature of an item interferes

with the successful running of a plan.

7.4.1. DF:DISABLED-CONDITION (DF:DC)

DF:DC describes situations in which a desired feature of an object causes a step to have a bad result.

The features used to recognize this situation are:

- There is a failure.
- There is a violated condition on the step that caused it.
- The plan to achieve the desired state has a violated satisfaction condition.
- The violating state satisfies at least one goal directly or indirectly.
- The condition is an inherent feature of an object.

There are three repairs associated with this TOP:

ADJUNCT-PLAN:PROTECT,
ALTER-PLAN:PRECONDITION,
ALTER-FEATURE.

7.4.2. DF:BLOCKED-PRECONDITION (DF:BP)

This TOP describes a situation in which an object's feature blocks a step but the feature itself is a desired one. Because the step is actually blocked by the feature, thus removing the possibility of running an adjunct plan as in DF:DC, the strategies associated with this TOP are very limited.

The features used to recognize this situation are:

- There is a blocked precondition.
- The condition that caused the block is a feature of an object.
- The condition satisfies at least one goal directly or indirectly.

The strategies that this TOP organizes are again aimed at changing the violated step itself and altering the feature of the object involved:

ALTER-FEATURE,
ALTER-PLAN PRECONDITION.

7.5. STEP-PARAMETER (SP)

The last family of TOPs relates to problems involving the amount of time a step takes and the tools it uses. The STEP-PARAMETER family has two TOPS:

- STEP-PARAMETER:TIME: a failure is blamed on the amount of time a step has run.
- STEP-PARAMETER:TOOL: A failure is blamed on the tool used in a step.

SP TOPS describe situations in which the bad parameter of a step interferes with the successful running of a plan

7.5.1. SP:TIME (SP:T)

SP:T is actually two TOPS, one for steps that have run too long and one for steps that have not run long enough. These are particularly useful TOPS in the CHEF domain where cooking times have a great effect on the results of a step. While significant, however, the features used to index to the TOPS and strategies used to repair the problems described by those TOPS are trivial.

The indices for SP:T:LONG, which describes situations in which steps have been run too long, are:

- There is a failure.
- The plan to achieve the desired state has a violated satisfaction condition.
- The violated condition is a time that is greater than the time constraint on the objects of the step.

There are two strategies to deal with this problem:

SPLIT-AND-REFORM,
ALTE R-TIME: D OWN.

The indices for SP:T:SHORT, which describes the reverse situation, in which a step has not been run long enough, are:

- There is a failure.
- The plan to achieve the desired state has a violated satisfaction condition.
- The violated condition is that the time on the step is less than the time required by the objects of the step.

Again there are two strategies to deal with this problem:

ALTER-TIME:UP,
SPLIT-AND-REFORM.

7.5.2. SP:TOOL (SP:TL)

Sometimes the instruments used in a step lead to problems of their own. Baking a souffle in the wrong size pan. for example, will lead to the outside cooking too fast or too slowly. SP:TL describes situations in which a step fails because of an inappropriate instrument.

The indices for SP:TL are:

- There is a failure.
- The instrument used causes the failure.

There is only one strategy associated with this TOP:

ALTER-TOOL.

7.6. Why TOPs?

In CHEF, TOPs are used to organize the repair strategies that can be applied to the problems that they describe. Strictly speaking, this is not necessary. The different strategies could each be indexed under the particular causal connection that they deal with. Instead of using a causal description of a situation to index to a TOP and then accessing the strategies that are housed under it, a planner could use the pieces of the same causal description to index directly to each of the strategies independently. Given this, what function do CHEF's TOPs really serve?

The answer to this has three parts: two within the CHEF implementation and one outside of it.

The first part of this answer is that CHEF uses its TOPS for something other than organizing repair strategies. It also uses them to guide it in learning from its failures. Along with the strategies it organizes, each TOP also indicates which features in the situation it describes should be marked as predictors of the current problem. Because each TOP corresponds to a particular causal configuration, it can store the information as to which aspects of the overall situation are important for CHEF to notice in future planning.

For example, CHEF had to repair the problem of the iodine taste of fish ruining a dish. In doing so. it diagnosed the situation as one of SIDE-FEATURE:ENABLES-BAD-CONDITION. This TOP organizes a set of strategies that can be applied to the problem, but because the TOP describes a situation in which a feature of an item interacts with a step (the taste of sea food and the STIR-FRY step), it is able to direct the planner to mark both of them as predictive of the failure. Once this is done, the co-occurrence of the two features, the goal to include sea-food and the goal to make a stir fry dish, will cause the planner to anticipate and try to avoid the failure. Neither feature alone, however, predicts this problem.

The TOP SIDE-FEATURE:GOAL-VIOLATION, which describes situations in which a feature of an item directly violates a goal, directs the learning in a slightly different direction. Like SF:EBC, SF:GV states that the feature of the object should be marked, but it does not suggest that any particular step be marked, because the feature violates one of the planner's goals independent of any step that has acted on it. CHEF diagnoses the greasy texture in the duck dumplings it builds as a case of SF:GV, because duck will have fat in any situation. As a result, CHEF marks the duck itself as predictive of a greasy-texture in all circumstances.

CHEF uses its TOPS to indicate which aspect of a situation should be marked as predictive of a problem in later planning. Because the TOP corresponds to the causal circumstances of the situation, it can indicate which features are important to a current problem and which will be important to later ones. Unlike organizing repair strategies, this is a function that requires the knowledge of all aspects of a planning problem. The entire description of the problem is needed to choose the features that will be marked. Even if the individual aspects of the problem could be used independently to point out individual features that were important, they would have to be combined at some point to form the proper conjunction of features that would predict a particular problem. One way or another, the configurations that correspond to the TOPS would have to be recognized.

The second answer to the question of the functionality of TOPS has to do with the prediction of failures and the indexing of plans. When CHEF anticipates a problem it has to have some way of describing that problem to its RETRIEVER so it can search for a plan that solves it. Instead of using a description that stays at the level of a specific failure (e.g.. soggy broccoli), CHEF uses its TOPS to describe what it predicts and what it is looking for. CHEF also indexes the plans that deal with problems by the TOPS that described them. The prediction of a problem described by a TOP is used to find a plan that solves that problem. The TOPs allow the planner to access plans that solve problems that are analogous to current ones.

The different plans that are indexed by a single TOP are similar in that the causal configuration of the problems they solve are the same. Each TOP is used to index to the plans that deal with different

instances of the single causal situation it describes. So each TOP acts as the access mechanism that links problems to the analogous plans in memory that will solve them.

One possibility that CHEF does not explore in dealing with plan failures is searching for another plan to replace its current one altogether. This would be a process similar to the initial retrieval of a plan to deal with a problem that CHEF does when it predicts one. In this case, however, it would happen after CHEF actually encounters a failure and has to recover from it. In this sort of situation, the TOPS themselves could be used to store past episodes that deal with the problems they describe. This situation demands that the plan being searched for deals with the causality of the current problem, which is described by the TOP, and so it makes sense to store these plans under the TOP itself. TOPs could carry greater weight by directly storing complete replacement plans to deal with failures as well as strategies.

Beyond this, TOPS could also be used to help deal with the problems that would rise out of a more realistic implementation of repair strategies. In a more ambitious implementation of strategies, one in which the table look-up of steps that meet certain requirements is replaced with a planner that can deal with simple state changes, the current approach to the competition between strategies would have to change. Generating all possible alterations and then choosing between them is too costly. A better approach would be to associate the successful and unsuccessful applications of a strategy to problems described by a TOP with the TOP itself. These episodes could then *focus* the planner's attention on the strategies that have been most helpful in this situation, *suggest* implementations of different strategies, and *warn* if certain strategies have had problematic results in this type of situation [6]. These functions have to be linked to the TOPS in that the competition between strategies has to be between them as they apply to the current problem, not as they apply globally to all problems.

In CHEF, TOPs organize strategies, indicate the features that will predict a failure, and link predictions of problems to the plans that solve them. They also have the potential to organize plans so that they can be suggested as replacements for failed plans and to manage better the choice between strategies that can be applied to a problem. While many of these functions could be implemented without the use of TOPS, others require TOPS because they correspond to entire causal situations and can be used to organize inferences that have to be made about those situations. The functional gains from the use of TOPs more than justifies their presence as organizing structures for planning.

8. The Repairs

The repair strategies used by CHEF owe a great deal to the work on plan repair that has preceded it, such as that of Sacerdoti [16]. Sussman [24], Tate [25] and Wilensky [28]. CHEFs repair rules, however, are somewhat more detailed than those that have gone before and make greater use of an organization that links the description of a problem to the solutions that can be applied to it.

Because many planning projects have been developed in somewhat impoverished domains many repair strategies have been overlooked because they had no application within these domains. Sussman's blocks world and Sacerdoti's pump world limited the actions of any planner working within them and thus limited the kinds of repairs that a planner could make. The failures encountered by HACKER and NOAH were usually the product of scheduling errors and were repaired by one or another variation of REORDER. They did not, in general, have to deal with questions such as the difference between a side effect and a desired effect of an action. Wilensky suggested a wider range of plan repair strategies in PANDORA, but did not provide the organization that would have made them useful or give a method for actually implementing the general strategies for use when solving specific problems.

> CHEF'S repair strategies differ from those of past planners in that they are more specific and are organized such that a single situation can have multiple strategies applied to it.

In CHEF the repair rules are all organized under specific TOPS and each TOP is associated with a particular causal configuration. The TOP SIDE-EFFECT:DIS-ABLED-CONDITION:CONCURRENT, for example, is accessed only when a single action has a side effect that disables states that have to be maintained during the course of that action. CHEF accesses these strategies by accessing the TOP that stores them, and the TOP is indexed by the causal explanation of the problem that corresponds to it.

For CHEF it is always the case that the repair strategies that are applied to a particular problem are those and only those that suggest changes to the causal configuration that will repair the plan. It may be that the change is not possible, because of the specifics of the problem, but if the change can be made then the plan will be repaired. In the beef and broccoli failure, one of the suggested repairs is to run an adjunct plan that will remove the liquid from the pan as the broccoli is cooking. In this case, CHEF does not know of any step to do this, but if such a step were known, such as throwing a sponge in the pan with

the food, it would solve the problem. Even though it cannot find such a plan, CHEF is able to describe the kind of action it needs, in terms of the preconditions and effects of different steps.

In order to use this description to find specific plans CHEF makes use of a library of actions, indexed by their preconditions and effects and by other plans that they are similar to. When a repair strategy is applied. CHEF builds a request into this library using the general description associated with the strategy, filled in with the specifics of the problem. The general strategy is turned into a specific request for a step or set of steps in the domain that have particular preconditions and effects.

Requests to this library of steps can take two forms. They can be requests for steps that have particular results or requests for steps that are similar to given steps but have different preconditions or effects. A strategy such as ALTER-PLAN:SIDE-EFFECT, that suggests replacing an existing step with one that does not produce a particular side effect, builds a request of the second type (Fig. 12). A strategy such as RECOVER, that directs the planner to add a new step to a plan that will remove a particular state, builds a request of the first type (Fig. 13). Each one is defined as a general structure which is filled in with the current answers to CHEF's explanation questions.

For example, one problem that CHEF encounters in creating a plan for FISH-WITH-PEANUTS is that the rice wine marinade it uses to remove the iodine taste of the fish makes the fish crumble during cooking. Because it recognizes that the side effect of one plan is enabling an undesired condition, CHEF evokes the TOP SIDE-EFFECT:DISABLED-CONDITION:SERIAL, which suggests the Strategy ALTER-PLAN:SIDE-EFFECT, as well as many Others. ALTER-PLAN:SIDE-EFFECT is a general strategy to use a plan that achieves the initial goal but does not have the undesired side effect. Because CHEF knows the problematic state, the goal that is to be achieved, and the initial plan that has gone wrong, it is able to build a request for a step described by the general strategy. It builds a request for a plan that removes the iodine taste from fish that does not have the side effect of adding liquid to the dish. This describes the plan segment in which the fish is marinated in ginger.

```
(def:strat alter-plan:side-effect
 bindings (*condition* expanswer-condition
           *step* expanswer-step
           *sgoal* expanswer-step-goal)
 question (enter-text ("Is there a alternative to " t
                        *step* t
                       "that will enable " t
                        *sgoal* t
                       "that does not cause " t
                        *condition*)
           test (search-alter-memory nil *step* *condition* *sgoal*)
           exit-text ("There is a plan" *answer*)
           fail-text ("No alternate plan found")
 response (text ("Response: Instead of doing step: " *step* t
                 "      Do: " *answer*)
           action (replace *step* *answer*))))
```

Fig. 12. Definition of ALTER-PLAN:SIDE-EFFECT strategy.

```
(def:strat recover
 bindings (*condition* expanswer-condition
           *step* expanswer-step)
 question (enter-text ("Is there a plan to recover from "
                        *condition*)
           test (search-step-memory *condition* nil)
           exit-text ("There is a plan" *answer*)
           fail-text ("No recover plan found"))
 response (text ("Response: After doing step: " *step* t
                 "          Do: " *answer*)
           action (after *step* *answer*)))
```

Fig. 13. Definition of RECOVER strategy.

CHEF uses its explanation of what has gone wrong to suggest the TOP, which "n turn -suggests a set of possible repairs. CHEF then uses the explanation again to provide the specific states and steps that convert the general repair rules into domain-specific requests for plans. Having a detailed causal description of the problems it encounters allows it to find very general repair rules and instantiate them as specific repairs.

To apply an abstract strategy, CHEF fills in the general structure with specific

information about the current situation from its explanation of the problem.

CHEF uses seventeen general repair rules in the normal course of its planning. Each one of these is associated with one or more TOPS and suggests a fix to a specific causal problem. Each one of these rules carries with it a general description of a fix to a plan, through a reordering of steps, an alteration of the objects involved or a change of actions. These general descriptions are filled in with the specific states that the planner is concerned with at the time the repair rule is suggested.

It is important to realize that these strategies, while they are designed for specific situations, are not *domain*-specific. Stated generally, they are:

- 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.
- ALTER-PLAN:PRECONDITION: Replace the step with the violated precondition with one that does not have the same precondition but achieves the same goal.
- RECOVER: Add a step that will remove the side effect before the step it interferes with is run.
- REORDER: Reorder the running of two steps with respect to each other.
- ADJUST-BALANCE:UP: Increase the down side of a violated balance relationship.
- ADJUST-BALANCE:DOWN: Decrease the up side of a violated balance relationship.
- ADJUNCT-PLAN:REMOVE: Add a new step to be run along with a step that has the result of removing a problematic state as it is being created.
- ADJUNCT-PLAN:PROTECT: Add a new step to be run along with a step that is affected by an existing condition that allows the initial step to run as usual.
- ALTER-TIME:UP: Increase the duration of a step.
- ALTER-TIME:DOWN: Decrease the duration of a step. -ALTER-ITEM: Replace an existing item with one that has the desired
- features but not an undesired one.
- ALTER-TOOL: Replace an existing tool with one that has the desired effect but does not cause an undesired one.
- SPLIT-AND-REFORM: Split a step into two separate steps and run them independently.
- ALTER-PLACEMENT:BEFORE: Move an existing step to run before another one.
- ALTER-PLACEMENT:AFTER: Move an existing step to run after another one.
- ALTER-FEATURE: Add a step that will change an undesired attribute to the desired one.
- REMOVE-FEATURE: Add a step that will remove an inherent feature from an item.

Each of these strategies is associated with one or more planning TOPS. Each TOP is indexed by a general description of the type of plan failure that its strategies can repair. This allows the explanation of a failure to be used to access the TOP that contains the strategies that can repair it.

9. A Causal Vocabulary for TOPs

Each of CHEF'S TOPS corresponds to a causal configuration that describes a failure. The vocabulary that is used to build up these configurations and define the TOPS is one of causal interactions between steps and states. This vocabulary differs from those used by past planners to describe the problems they encounter primarily in that it distinguishes the effects of steps that lead to satisfied goals from those that do not. It also distinguishes problems that cause a plan to halt from those that allow the plan to run to completion but ruin the result.

CHEF'S causal vocabulary is based on four classes of things that it knows about:

(1) OBJECTS, which are physical objects such as the ingredients and the utensils used in the CHEF domain.
(2) STATES, which are features of the OBJECTS such as TASTE, TEXTURE and SIZE.
(3) STEPS, which are actions taken on OBJECTS with the aim of altering some STATES.
(4) GOALS, which are STATES that the planner is trying to achieve. Unlike other STATES, GOALS are hypothetical STATES that actual STATES are tested against.

With the exception of a distinction between certain types of STATES, the bulk of CHEF'S vocabulary has to do with relationships between these OBJECTS, STATES and STEPS. One important aspect of the relationships that CHEF uses to describe causal configurations is that they depend on the presence of goals. Many of the relationships change as the planner's goals change, even if the causality of the situation does not. If a STATE satisfies a goal, that fact is reflected in the description of any problems that result from it. If the goal it satisfies is no longer part of the planner's overall goal structure, the description of the situation will change to reflect this. CHEF'S descriptions of causal situations depend on its goals because it has to use these description to find appropriate changes to make to its plans, and these changes cannot be

such that they violate the goals already met by these plans.

The vocabulary used to describe planning TOPS is based on the different relationships that can hold between OBJECTS, STATES, STEPS and GOALS.

One of the most powerful distinctions that CHEF'S vocabulary makes use of is the distinction between different relationships between STATES and the STEPS that cause them. CHEF understands three types of relationships:

(1) DESIRED-EFFECTS are states that result from steps that satisfy the planner's goals. This satisfaction can either be the direct result of the state matching a goal state or the indirect result of the state enabling a later step that satisfies a goal.
(2) SIDE-EFFECTS are the results of steps that do not satisfy any goals either directly or indirectly.
(3) DEVELOPING-SIDE-EFFECTS are side effects that occur during the running of a step rather than as a final result.

All of the effects of a step are linked to it by RESULT links that lead from the step to the state and RESULT-OF links that lead from the state back to the step.

The effect of distinguishing between these different relationships lies in the types of strategies that each will allow. Problems having to do with SIDE-EFFECTS can be solved by making changes to a plan that remove the effects in one way or another while those having to do with DESIRED-EFFECTS cannot. Likewise, those states that are DEVELOPING-SIDE-EFFECTS can be dealt with using adjunct steps that can be added to a plan to remove the effects of the steps as they are produced.

CHEF recognizes three types of preconditions for STEPS:

(1) REQUIRED-PRECONDITIONS are those states that have to hold for a step to be run at all.
(2) SATISFACTION-CONDITIONS are those states that have to hold as a step is entered for its desired effects to result.
(3) MAINTENANCE-CONDITIONS are those states that have to hold during the entire running of a step for its desired effects to result.

STATES that meet the preconditions of a STEP have ENABLES links going from the STATES to the STEP. The STEP has ENABLED-BY links going from it back to the STATES. For example, the STATE representing the texture of the crushed strawberries in a strawberry souffle plan is linked to the MIX step by an ENABLES link and the MIX step points back to it by an ENABLED-BY link.

Along with the relationships between STATES and STEPS there are relationships between OBJECTS and STATES and between OBJECTS and STEPS. An OBJECT can link into a STEP in two ways:

(1) As a STEP-OBJECT, which means that the STEP acts on the OBJECT in order to change a STATE related to it.
(2) As an INSTRUMENT, which means that the STEP uses the OBJECT in order to enable a STATE change in another OBJECT.

The STATES that CHEF knows about are actually features of OBJECTS. The different STATES describe the TASTE, TEXTURE and SIZE of the OBJECTS as they are modified by the STEPS in the plan. Each STATE is not only the RESULT of a STEP, it is also a feature of an object. The texture of strawberries after they are pulped is the RESULT of the PULP step but it is also a feature of the strawberries.

CHEF only recognizes two types of relationships between an OBJECT and the STATES that define its features:

(1) DESIRED-FEATURE is a feature of an OBJECT that satisfies or enables the satisfaction of a goal.
(2) SIDE-FEATURE is a feature that is incidental to the satisfaction of any current goals. It is not UNDESIRED in that it does not necessarily cause problems. It just does not satisfy any goals.

There are three relationships that a STATE can have with a GOAL:

(1) A SATISFY relationship exists when one STATE matches the preconditions of a step or the final GOAL of a plan.
(2) A VIOLATE or BLOCK relationship exists when a particular STATE of an object is supposed to SATISFY a GOAL or step precondition but does not.
(3) A DISABLE relationship exists when a state VIOLATES the satisfaction conditions of a step.

STEPS can only relate to one another in terms of their running order in the plan. CHEF only requires two relationships to describe its failure situations:

(1) Two STEPS are SERIAL when they are run at different times.
(2) Two STEPS are CONCURRENT when they are run at the same time.

Like other features that CHEF uses to describe situations, the time sequence of two STEPS determines the strategies that can be applied to different problems involving the interactions between them.

The last set of features that CHEF uses to describe situations has to do with nature of the STATES it has to deal with rather than the relationship between STATES and either STEPS or OBJECTS. The distinction here, however, still makes use of the notion of "relationship" in that it points to the fact that some states are themselves relationships between many STATES at once. CHEF understands two sorts of STATES:

(1) A BALANCE STATE is defined as a relationship that has to hold between two or more STATES.

(2) An ABSOLUTE STATE is the feature of a single OBJECT.

By noticing this distinction CHEF is able to suggest strategies in cases of violated BALANCE STATES that it is not able to suggest in cases of violated ABSOLUTE STATES.

The vocabulary of TOPS is one of causal interactions. It describes the relationships between OBJECTS, STATES, STEPS and GOALS and makes distinctions between those that serve goals and those that do not. It distinguishes between these different TOPS because the different problems described by them require different repair strategies.

10. Repairs Not Used by CHEF

While CHEF uses quite a few repair strategies to deal with its planning failures, there exists a whole set of these repair rules that it ignores. CHEF excludes these strategies not because they are not useful, but because they deal with problems that CHEF never encounters and suggest changes that make no sense in CHEF'S domain. For example, CHEF does not have strategies to deal with planning failures having to do with interactions with other actors. The entire range of *counter-planning* strategies suggested by Carbonell [2] and Wilensky [26] that were designed to deal with problems that arise when two or more planners interact have been ignored. CHEF also ignores those strategies that require that it alter the initial goals that it is handed as its input. While the changes that CHEF makes to a plan may modify goals that it has generated on its own, it never makes any changes that require modifications of the initial goals it is given as input. This means that subgoals that CHEF is planning for may be changed, but the main goals that CHEF has remain intact.

CHEF is not able to delegate its planning tasks to others, so it does not use strategies such as ALTER-ACTOR, because the problem that this strategy solves never arises for CHEF. Likewise, although the actions controlled by the planner all have durations which can be changed, none of them have alterable rates. For example, it is not possible for CHEF to change the amount of time that it takes to chop vegetables. It can change the duration of cooking steps, but it cannot change the rates of those steps. This means that it cannot use strategies such as SPEED-UP or SLOW-DOWN. They are useful when the amount of time that an action takes to perform can be altered by the planner by changing the rate at which he performs them. These strategies allow a planner to coordinate the times when actions end, to minimize ill effects produced by the performance of an action, or maximize any positive effects produced.

In general, domains with different sorts of actions will allow a planner to use different sorts of strategies. Just as the cooking domain allows CHEF to make use of more interesting and powerful strategies than are possible in a blocks world, a more expansive domain than cooking would suggest a greater set of strategies than CHEF could make use of. The main reason for the expansion of strategies in CHEF, however, has less to do with the domain and more to do with the addition of the vocabulary required to distinguish between goal satisfying and non-goal satisfying actions and states.

11. Planners and Repair

CHEF's approach to failure recovery draws a great deal from the tradition established by Sussman [24] and Sacerdoti [16] in their use of planning critics, and then extended by Wilensky [28] in PANDORA. Both Sussman and Sacerdoti suggested that planners need to know about planning errors in general to debug faulty plans intelligently. With this knowledge, a planner would not have to perform the backtracking that a STRIPS-like planner must do when it hits a dead end. Instead, it would be able to apply one of a set of planning critics, each having knowledge of a specific problem and the means to solve it, thus repairing the faulty plan by changing only the steps that participate in the failure. The primary difference between the two views of planning critics suggested by Sussman and Sacerdoti had to do with when the critics were applied. Sussman's HACKER would apply its critics while running a fully expanded plan in a careful mode while Sacerdoti's NOAH would have its critics check for plan interactions at each stage of expansion down to primitive planning steps. Sussman's approach required the assumption that the plan was linearized, while Sacerdoti's required complete knowledge of the effects of all of the actions in a domain.

There are difficulties with the critics approach presented by both Sussman and Sacerdoti, however. The first stems from the fact that they both wrote planners that functioned in somewhat impoverished

domains, Sussman working in a blocks world and Sacerdoti working in a world where the problem task was the building of simple machines such as pumps. Unfortunately, in such worlds, there is usually only one plan for each goal and the effects of each plan tend to be those and only those that directly satisfy the goal being planned for. Because of this, only a limited vocabulary was needed to describe the problems the critics were confronted with and each problem required only a single transformation to solve it. As a result, the critics suggested by both Sussman and Sacerdoti only begin to scratch the surface of what is needed to deal with complex domains.

In an effort to remedy these problems with critics, Wilensky suggested the idea of *metaplanning* [28]. Rather than representing abstract planning knowledge in a set of critics, Wilensky represented it in the form of metagoals and metaplans that are identical in form to the specific goals and plans of the planner's domain. When confronted with a problem stemming from plan interactions, a goal to deal with that interaction is spawned and the general planning mechanism deals with it directly. This allows the planner to notice a problem, insert a goal to deal with it into its general planning agenda, and then find plans to deal with it. Wilensky has taken the same knowledge that Sussman and Sacerdoti stored in critics, and has divided it into diagnostic knowledge which recognizes problems and prescriptive knowledge which solves them.

In building his planner, Wilensky argued that Sussman and Sacerdoti did not go far enough in taking seriously the role of goal and plan interactions, the main source of planning failures. He argued that the way to deal with problems rising out of goal and plan interactions is to elevate them to the status of goals themselves. He argued, in fact, that the basic structure of the planner should reflect the fact that it will often be encountering failures due to these interactions. This argument is continued in CHEF, and extended to include the argument that a planner needs to have a rich causal description of *why* a failure has occurred rather than just a description of *what* the failure is.

More recently, Chapman [4] has described a planner, TWEAK, that uses a simple repair strategy that grows out of a formal analysis of when a statement can be understood as provably valid in the context of a nonlinear planner. The purpose of Chapman's approach to repair, and to TWEAK in general, was not to build a practical system or to make points about repair, but was instead to make formal claims about the complexity of a provably correct and complete planner. The approach taken in CHEF, while by no means complete, is aimed at the more practical problem of finding an effective solution to a local planning failure that will be less likely than others to introduce new problems. There is a further difference in that TWEAK is concerned solely with problems in domains that can be fully described using STRIPS operators. As a result, TWEAK lacks a full notion of time or repairs that have to do with time. Many of CHEF'S repair rules, however, have to do with the problems of overlapping and simultaneous actions as well as states that develop over the duration of an action.

There is little sense, however, in comparing CHEF and TWEAK in that TWEAK exists primarily to point out the complexity problems in nonlinear planning while CHEF is an attempt to avoid some of them. In particular, CHEF avoids the problem of the complexity of projection by replacing exhaustive projection with a heuristically driven process of anticipation that is based on the planner's memory of past failure rather than its full knowledge of the physics of a domain.

Another approach to repair has been suggested by Wilkins in the SIPE planner [30]. Like CHEF, SIPE is designed to repair problems detected at execution that for one reason or another were missed at planning time. Like CHEF, SIPE makes use of a taxonomy of planning problems and repairs that is used to drive what Wilkins refers to as "replanning." Unlike CHEF, however, SIPE's primary replanning strategies center around identifying problematic states and handing descriptions of the state and the desired world back to the planner. In effect, SIPE responds to failures by actually replanning rather than repairing. There is little to be gained in SIPE by building an explanation of the failure, in that no retroactive changes can be made to the plan.

For many types of problems, this strategy is a very effective one, but it is not the best when the actual goal of a system is not just to repair the plan in place, but also to save and reuse the corrected version. Rather than changing a plan so as to reinstate any missing states, CHEF attempts to change the plan in a way that allows later execution of the plan to avoid the problem altogether. In this way, the plan is required for reuse rather than continuation. More recently, we have been concerned with both aspects of this problem – continuation of a current plan as well as repair of the plan for later reuse.

12. Conclusions

By explaining plan failures, CHEF is able to make use of a broad range of plan repairs that a less informed system would not be able to apply reliably. The explanation gives the planner the knowledge it

needs to choose those and only those repairs that will fix a plan without introducing new problems to it. As a result, strategies with greater power but less general applicability can be used with confidence. Furthermore, by dividing the task of plan repair into diagnosis and treatment, the system has the flexibility to try multiple strategies for repairing a single failure and choose the one most appropriate for the particular problem. The method as a whole gains in both range and power. Its repair strategies are applicable to a wide range of planning problems, while each strategy is itself a specific and thus powerful repair.

REFERENCES

1. R. Alterman, Adaptive planning: Refitting old plans to new situations, in: *Proceedings Seventh Annual Conference of the Cognitive Science Society,* Irvine, CA (1985).
2. J.G. Carbonell, Subjective understanding: Computer models of belief systems, Ph.D. Thesis, Yale University, New Haven. CT (1979).
3. J.G. Carbonell, A computational model of analogical problem solving, in: *Proceedings IJCAI-81,* Vancouver, BC (1981).
4. D. Chapman, Planning for conjunctive goals. *Artificial Intelligence 32* (1987) 333-377.
5. R. E. Fikes and N.J. Nilsson, STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence 2* (1971) 189-208.
6. K.J. Hammond, Indexing and causality: The organization of plans and strategies in memory. Tech. Rept. 351, Yale University Department of Computer Science, New Haven, CT (1984).
7. K.J. Hammond, Learning and re-using explanations, in: *Proceedings Fourth International Conference on Machine Learning.* Irvine, CA (1987).
8. K.J. Hammond, *Case-Based Planning: Viewing Planning as a Memory Task* (Academic Press, New York, 1989).
9. K.J. Hammond and N. Hurwitz, Extracting diagnostic features from explanations, in: *Proceedings 1988 AAAI Symposium on Explanation-Based Learning* (1988).
10. J.L. Kolodner and R.L. Simpson, Experience and problem solving: A framework, in: *Proceedings Sixth Annual Conference of the Cognitive Science Society,* Boulder, CO (1984).
11. J.L. Kolodner, R.L. Simpson and K. Sycara, A process model of case-based reasoning in problem-solving, in: *Proceedings IJCAI-85,* Los Angeles, CA (1985).
12. P. Koton, Using experience in learning and problem solving. Ph.D. Thesis. MIT. Cambridge, MA (1988).
13. C. Martin and C.K. Riesbeck, Uniform parsing and inferencing for learning, in: *Proceedings AAAI-86.* Philadelphia, PA (1986).
14. D. McDermott. Planning and acting, *Cogn. Sci. 2* (1978) 71-109.
15. E. Rissland and K. Ashley. Hypothetical as heuristic device, in: *Proceedings AAAI-86,* Philadelphia, PA (1986).
16. E.D. Sacerdoti. A structure for plans and behavior. Tech. Rept. 109. SRI Artificial Intelligence Center. Menlo Park, CA (1975).
17. R.C. Schank, *Dynamic Memory: A Theory of Learning in Computers and People* (Cambridge University Press. Cambridge, 1982).
18. R.C. Schank, *Explanation Patterns: Understanding Mechanically and Creatively* (Erlbaum, Hillsdale, NJ, 1986).
19. R.C. Schank and R. Abelson, Scripts, plans, and knowledge, in: *Proceedings IJCAI-75,* Tbilisi, USSR (1975).
20. C. Seifert, G. McKoon. R. Abelson and R. Ratciiff. Memory connections between thematical-ly similar episodes. *J. Experimental Psychol. Learning Memory Cogn.* (1985).
21. R.F. Simmons and R. Davis, Generate, test, and debug: Combining associational rules and causal models, in: *Proceedings IJCAI-87,* Milan, Italy (1987).
22. R.L. Simpson, A computer model of case-based reasoning in problem solving, Ph.D. Thesis, Georgia Institute of Technology. Atlanta. GA (1985).
23. M.J. Stefik. Planning and meta-planning (MOLGEN: Part 2), *Artificial Intelligence* 16 (1981) 141-169.

24. G.J. Sussman, *A Computer Model of Skill Acquisition.* Artificial Intelligence Series 1 (American Elsevier. New York. 1975).
25. A. Tate, Generating project networks, in: *Proceedings IJCAI-77,* Cambridge, MA (1977).
26. R. Wilensky, Understanding goal-based stories. Ph.D. Thesis. Research Rept. #140. Yale University, New Haven, CT (1978).
27. R. Wilensky. Meta-planning: Representing and using knowledge about planning in problem solving and natural language understanding. *Cogn. Sci. 5* (1981).
28. R. Wilensky, *Planning and Understanding: A Computational Approach to Human Reasoning* (Addison-Wesley, Reading. MA, 1983).
29. D.E. Wilkins. Domain-independent planning: Representation and plan generation. *Artificial Intelligence 22* (1984) 269-301.
30. D.E. Wilkins, Recovering from execution errors in SIPE. *Compui. Intell.* 1 (1985) 33.

Received November 1988; revised version received September 1989