Explaining and Repairing Plans that Fail

Kristian J. Hammond

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

ABSTRACT

A persistent problem in machine planning is that of repairing plans that fail. Two solutions have been suggested to deal with this problem: planning critics and met a-planning techniques. Unfortunately, both of these suggestions suffer from lack of flexibility due to an extremely restricted view of how to describe planning failures.

This paper presents an alternative approach in which plan failures are described in terms of causal explanations of why they occurred. These explanations are used to access different abstract replanning strategies, which are then turned into 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 Szee hwan cooking.

I. THE PROBLEM OF PLAN FAILURE.

All planners face the problem of plans that fail. As a result, most planner's have some mechanism for plan repair or debugging. These have ranged from simple backup and replanning algorithms [14] to complex meta-planning techniques [17]. Unfortunately, all of these mechanisms have suffered from a very restricted view of what is involved in describing a failure. They describe failures in terms of the individual states that define them but are not concerned with the causes of those states. Because of this, plan repair techniques have been limited to local changes that do not make use of the range of responses that a more global understanding of *why* a failure has occurred would allow.

This paper presents a different kind of 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 is embedded in the computer program CHEF, a case-based planner that creates new plans in the domain of Szechwan cooking. When CHEF encounters a failure in one of its own plans, it explains why the failure has occurred, using a set of causal rules concerning the effects of the actions in its domain under different circumstances. 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 index to a set of abstract repair strategies appropriate to the general description of the problem. CHEF then tries to implement the different strategies and chooses the one best suited for the specific problem. Because the planner has a detailed description *of* what has gone wrong in a situation, it is able to find and make use of strategies that are less general purpose but are more powerful in the specific situations.

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

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 [2,11,15,16]. In this case, the problem takes the form of choosing which plan repair, among many, should be applied to a faulty plan. The goal is to have a planner that is able to diagnose its own failures as being one type or another and then use this diagnosis to choose and apply the change (or repair strategy) that will actually result in a correct plan. The approach described in this paper, one which also reflects the approach taken by human planners, 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 knows about.

II. AN OVERVIEW OF CHEF,

CHEF is a case-based planner that, like other case-based reasoning systems [1,3,4,6,7,9,10], builds new plans out of its memory of old ones. CHEF's domain is Szechwan cooking and its task is to build new recipes on the basis of a user's requests. CHEF's input is a set of goals for different tastes, textures, ingredients and types of dishes and its output is a single recipe that satisfies all of its goals. Its basic algorithm is to find a past plan that satisfies as many of the most important goals as possible and then modify that plan to satisfy the other goals as well.

Before searching for a plan to modify, CHEF examines the goals in its input and predicts any failures that might rise out the interactions between the plans for satisfying them. If a failure is predicted, CHEF adds a goal to avoid the failure it to it6 list of goals to satisfy and this new goal is also used to search for a plan. The power of CHEF lies in its ability to predict and thus avoid failures it has encountered before. The topic of this paper, however, is its ability to repair planning failures when it encounters them.

CHEF consists of six modules:

- The 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.
- The 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.
- The MODIFIER alters the plan found by the RETRIEVER to achieve any goals from the input that it does not satisfy.
- t The 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.

- The ASSIGNEE, uses the causal explanation built by the REPAIRER to determine the features which will predict this failure in the future.
- The STORER places new plans in memory, indexed by the goals that they satisfy and the problems that they avoid.

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 symbolically 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 the goals of a plan are handed to the REPAIRER for repair.

III. CHEF'S REPAIR ALGORITHM.

CHEF's process of failure repair has five phases to it:

- Notice the failure.
- Build a causal explanation of why it happened.
- Use the explanation to find a planning TOP with repair strategies.
- Apply each of the general repair strategies using the specifics of the problem.
- Choose and implement the best repair.

The reason behind most of these steps is 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 one of them to fix the plan. The only steps that are not quite as straightforward are the second and third steps of building an explanation and using it to find a Thematic Organization Packet [13] or TOP.

The explanation that CHEF builds for a failure is a causal description of why that failure occurred. These causal descriptions correspond to planning TOPs. These TOPs are planning versions of the understanding structures suggested by Schank to store information that relates to complex interactions of goals.

In CHEF, each TOP stores the repair strategies that will fix the faulty plans it describes. These TOP6 are not just descriptions of problems. They are descriptions of problems that are 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 the TOP deals with. To get to the strategies that will deal with a problem, then, 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.

```
(SIZE OBJECT (BEEF2) SIZE (CHUIK))
(SIZE OBJECT (BR0CC0LI1) SIZE (CHUMK))
(TEXTURE OBJECT (BEEF2) TEXTURE (TEIDER))
(TEXTURE OBJECT (BROCCOLI1) TEXTURE (SOGGY))
(TASTE OBJECT (BEEF2)
       TASTE (SAVORY IITEISITY (9.)))
(TASTE OBJECT (BROCCOLI1)
       TASTE (SAVORY IITEISITY (6.)))
(TASTE OBJECT (DISH)
       TASTE (AID (SALTY IITEISITY (9.))
                  (GARLICY IITEISITY (9.))
                  (SAVORY IKTEISITY (5.))))
```

Figure 1: Section of Plan Result Table.

iv. yfQTiqiNQ THE FAILVRE

After CHEF has built a plan, it runs a simulation of it. This simulation is the program's equivalent of the real world and a plan that makes it to simulation is considered to be complete. The result of this simulation is a table of statements that characterize the final states of the objects manipulated by the plan.

For example, 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 includes in the dish as a whole (figure 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. 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.

If a plan's goal is not found on the result table it generates when run, then the plan has not succeeded in achieving it and is considered a failure. This is the first kind of failure that CHEF can recognize. It is the failure of a plan to achieve one of its goals. Another type of failure is when a state is present on the table of results that is itself an objectionable state. These "objectionable" states include unwanted textures *{e.g.,* SOGGY broccoli and FLAT souffles) and undesirable tastes *(e.g.,* the iodine taste of fish and the oily taste of grease). The set of states that is considered objectionable is dynamic, changing in response to the different types of plans that are created. Some of these states are associated with particular types of dishes and only checked for when that type of dish is being made while others are more general and are always checked for. If one of these states is found on the table of results, the fact that it is there is also considered a failure of the proper running of the plan.

One example of a failure that is both a failure to satisfy a goal and the inclusion of a negative state is a problem that arises when CHEF is running a plan it has built for STRAWBERRY-SOUFFLE. To build this plan, CHEF has modified an existing VANILLA-SOUFFLE plan, adding strawberries to the original. Unfortunately, the strawberries add extra liquid, which creates

an imbalance between the liquid in the souffle batter and leavening. This causes the souffte to fall. This fact is recorded by the simulation of the plan, and CHEF picks up the failure by testing the plan against the goals it is supposed to achieve.

```
Checking goals of recipe -> STRAWBERRY-SOUFFLE

- Checking goal ->
It should be the cat* that: The batttr is risen.
    The goal: The batter is now risen.
    is not satisfied.
It is instsad ths cass that: The battsr is flat.

- Checking for negative features ->
Unfortunately: Ths battsr is now a bad texture.
In that: The batter is now flat.

Recipe -> STRAWBERRY-SOUFFLE has failed goals.
```

v. EXPLAINING THE FAILVRE

Once a failure has been recognized, CHEF sets upon the task of building a causal explanation of why it has occurred. CHEF needs this explanation because the causal descriptions of failures are used to access the strategies that can be applied to them. 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. And the best description in this case is a causal explanation of the problem.

To build its explanations, CHEF uses the trace left by the forward chaining of the simulator. 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. 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's movement through the causal network built up by the simulator is controlled by a set of explanation questions. 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. These questions are aimed at discovering the actual step that caused the failure, the conditions that had to be true for it to occur, the cause of those conditions and the goals that the various steps and states were trying to satisfy.

In the example of the failed strawberry souflte, CHEF starts with the fact that the batter has ended up flat and chains back through the steps and their results to find that the chopped strawberries are the actual cause of the problem. On the path to this discovery, it finds that the relationship between liquid and leavening that is needed to make the souffle rise is out of balance and that the goal being served by the the strawberries is that of having the overall dish taste like berries. All of these facts participate in choosing the abstract strategies that will be suggested to repair the plan.

```
ASKIIG TEE QUESTIOI: »What is the failure?'
AISVER-> The failure is: It is not the case that:
         The batter is now risen.

ASKIIG TEE QUESTIOI: 'What is the preferred state?'
AISWER-> The preferred state is:
         The batter is now risen.

ASKIIG TEE QUESTIOI: 'Whet was the plan to achieve
the preferred stats?[1]
AISWER-> The plan was:
         Bake the better for twenty five minutes.

ASKIIG TEE QUESTIOI: 'What were the conditions that
led to the failure?[1]
AISWER-> The condition was:
         There is an imbalance between the whipped
         stuff and the thin liquid.

  Only one aspect of the imbalance:
  There is an imbalance between the whipped stuff
  and the thin liquid is unexpected.

 The state:
  There is whipped stuff in the bowl from the total
  equaling 60 teaspoons,
 normally participates in the goal:
  The batter is now risen.

 Only the other aspect of the imbalance:
  There is thin liquid in ths bowl from the
  strawberry equaling 2.4 teaspoons
 is an unexpected condition.

 ASKIIG TEE QUESTIOI: 'What caused the conditions
that led to the failure?'
AISWER-> There is thin liquid in the bowl from the
         strawberry equaling 2.4 teaspoons,
        was caused by: Pulp the strawberry.

ASKIIG TEE QUESTIOI: 'Do the conditions that caused
the failure satisfy any goals?'
AISWER-> 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.

ASKIIG TEE QUESTIOI: 'What goals does the step that
caused the condition enabling the failure satisfy?'
AISWER-> The step: Pulp the strawberry.
         establishes the preconditions for:
         Nix the strawberry with the vanilla,
         egg white, egg yolk, milk, sugar, salt,
         flour and butter.
        This in turn lsads to ths satisfaction of:
         The dish now tastes like berries.
```

VI. INDEXING TO THE TOP

CHEF's repair strategies are all stored under planning TOPs, 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 fixes to the failed plans described by the TOP. These fixes are designed to repair the failure without interfering with the other goals in any plan that the TOP describes. CHEF has sixteen TOPs that correspond to different causal situations and store different repair strategies. These TOPs include structures such as SIDE-EFFECT:DISABLED-CONDITION:BALANCE and SIDE-FEATUREiENABLES-BAD-CONDITION. The strategies include changes such as REORDER steps, REMOVE condition, and SPLIT-AND-REFORM step.

CHEF uses the answers to each of its explanation questions as an 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.

There are three parts to each of CHEF's repair TOPs: the indices used to access it, the repair strategies stored under it and the features of the situation that it describes that are to be marked as predictive of the problem later on. In this paper, only the first two of these are important, the indices and strategies. The answers to the explanation questions CHEF has asked are used to find a TOP. The choice of a TOP is dependent on the causality of the problem and each TOP corresponds to a different causal situation. The strategies that are stored under a TOP are a reflection of this, in that they are those and only those strategies that can be applied to repair the problem described by the TOP. The causal description of a problem is used to access the TOP that corresponds to it and thus access the strategies that can be applied to solve it.

In the case of the strawberry souffle failure, the fact that the condition violated in the plan is a balance requirement between two amounts and the fact that the condition that causes the imbalance is a side-effect of a step that does not satisfy any goals are 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, different TOPs, with different strategies would have been found.

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 * MIL
The goals that the step causing the condition enables « The dish now tastes like berries.

Found TOP -> SE:DC:B
TOP -> SIDE-EFFECT:DISABLED-COIDITIOI:BALAICE
has 5 strategies associated with it:

ALTER-PLAI:SIDE-EFFECT ALTER-PLAI:PRECOIDITIOI
ADJUMCT-PLAI RECOVER
ADJUST-BALAICE:UP

SIDE-EFFECT:DISABLED-CONDITION
is recognized when one step for satisfying a goal has a side-effect which disables a satisfaction condition for a later step and this satisfaction condition is a balance requirement between two states. Each strategy under this TOP suggests an alteration to the initial plan that will cause a break in one part of this causal chain. 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 changes one link in the causal chain that leads to the failure. The strategies under SIDE-EFFECT:DISABLED-CONDITION:BALANCEare:

- 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 that will allow it to satisfy the goal even in the presence of the violation.
- RECOVER: Add a new step between the step that causes the side-effect and the step that it blocks that removes the violating condition.
- 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.

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, the TOP found would have been the more general SIDE-EFFECT:DISABLED-CONDITION structure that lacks the ADJUST-BALANCE:UP strategy. 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.

VII- 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 (figure 2) and fill it in with the specific states that it suggests recovery from. In this way a general strategy for repairing a plan becomes a specific change to the plan at hand.

```
(def:strat recover
 bindings
   (♦condition* expanswer-condition
    ♦step* expanswer-step)
 question
   (enter-text ("Is there a plan to recover from M
                ♦condition*)
    test (search-step-memory *condition* nil)
    exit-text ("There is a plan" *answer*)
    fail-text ("Mo recover plan found"))
 response
   (text ("Response: After doing step: " *step* t
          "          Do: " *answer*)
   action (after *step* *answer*)))
```

Figure 2: Definition of the RECOVER strategy

Each strategy has two parts: a test and a response. The test under each strategy determines whether or not the strategy has an implementation in the current situation. The response is the change that is suggested. In most cases, the results 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. 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.

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 possible alter. The tests and responses are actually empty frames that are filled with the specifics of the 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 is so it can 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. When the strategy is actually applied, the specific answers are inserted into the appropriate slots in the strategy structure.

In the example of the strawberry souffle, 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.

Applying TOP ->
SIDE-EFFECT:DISABLED-CONDITION:BALAHCE
to failure It is not the case that: The batter is now risen - in recipe BADSTRAWBERRY-SOUFFLE

Asking questions needed lor 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

Response: Instead of doing: Pulp the strawberry
do: Using the 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.

Response: 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.

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

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

Asking questions needed for evaluating strategy:
ADJUST-BALANCE

ASKING ->
Can we add more whipped stuff to
BADSTRAWBERRY-SOUFFLE

Response: Increase the amount of egg white used.

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 causal 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, CHEF must have a mechanism for not only generating these changes, but also choosing between them.

VIII. 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 something that is already in the recipe than something new."[1] 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."

Once a change is selected, CHEF actually implements the change using its procedural knowledge of how to add new steps, split steps into pieces, remove steps and add or increase ingredients.

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.

IX. THE REPAIRS

The repair strategies used by CHEF owe a great deal to the work on plan repair that has preceded it [12,14,15]. CHEF's 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.

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 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 when the repair rule is suggested.

These strategies 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 causes a side-effect that removes the side-effect as it is 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 ingredient with one that have 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 the 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.

X. 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 chose 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. Further, 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 then choose the one most appropriate for the particular problem. The method as a whole, then, gains in range of different strategies that can be applied and the power of individual strategies.

REFERENCES

[I] A Herman,R., Adaptive Planning: Refitting old plans to new situations, in *The seventh annual conference of the cognitive Science Society,* 1985.

[2] Alterman, R., An Adaptive Planner. *AAAI-86,* 1986, 65-69.

[3] Carbonell, J.G., Derivational analogy and its role in problem solving. *AAAI-83,* 1983, 64-69.

[4] Carbonell, J. G , A computational model of analogical problem solving. *IJCAI 7*, 1981.

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

[6] Hammond, K., *Case-based Planning: An integrated theory of planning, learning and memory.,* Ph.D. Thesis, Yale University, 1986.

[7] Hammond, K., Indexing and Causality: The organization of plans and strategies in memory., Yale Department of Computer Science Technical Report 351, 1985.

[8] Hammond, K., CHEF: A model of case-based planning., AAAI-86, 1986, 267-271.

[9] Kolodner, J. L. and Simpson, R. L., Experience and problem solving: a framework. *Proceedings of the ninth annual conference of the cognitive science society,* 1984.

[10] Kolodner, J. L., Simpson, R. L. and Sycara-Cyranski, K., A process model of case-based reasoning in problem solving., *IJCAI 9,* 1985.

[II] McDermott, D., Planning and Acting, *Cognitive Science 2,* (1978), 71-109.

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

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

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

[15] Wilensky, R., *Planning and Understanding,* Addison-Wesley, Reading, Mass, 1983.