1. Case-Based Reasoning Applied to Planning

Ralph Bergmann, Héctor Muñoz-Avila, Manuela Veloso, and Erica Melis

1.1 Introduction

Planning means constructing a course of actions to achieve a specified set of goals when starting from an initial situation. For example, determining a sequence of actions (a plan) for transporting goods from an initial location to some destination is a typical planning problem in the transportation domain. Many planning problems are of practical interest.

The classical generative planning process consists mainly of a search through the space of possible operators to solve a given problem. For most practical problems this search is intractable. Therefore, case-based reasoning can be a good idea because it transfers previous solutions rather than searching for it.

Since the space of possible plans is typically vast, it is extremely unlikely that a case base contains a plan that can be reused without any modification. First, modification has been addressed in CHEF (Hammond 1986), one of the first case-based planners. It retrieves cooking recipes and adapts them to the new problem by using domain specific knowledge. As experience has shown, however, this kind of adaptation in realistic domains requires a large amount of very specific domain knowledge and lacks flexibility.

Given that classical generative planning may engage in a very large search effort and pure case-based planning may encounter insurmountable modification needs, several researchers have pursued a *synergistic* approach of generative and case-based planning. In a nutshell, the case-based planner provides plans previously generated for similar situations and the generative planner is used as a source of modification. In this paper, we present four systems that integrate generative and case-based planning: PRODIGY/ANALOGY developed at the CMU, CAPLAN/CBC and PARIS developed at the University of Kaiserslautern, and ABALONE developed at the Universities of Saarbrücken and Edinburgh. These systems are domain-independent case-based planners that accumulate and use planning cases to control the search. In these systems, cases encode knowledge on why and which operators were used for solving problems. In our synergistic systems the workload imposed on the generative planner depends on the amount of modification that is required to completely adapt a retrieved case.

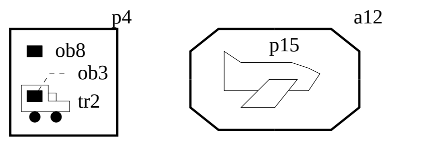
1.2 Generative Planning

Since the presented case-based planners are built on top of generative planners, we briefly introduce generative planning.

Together *initial state* and a set of *goals* form a *planning problem*. A planning task consists of finding a *plan*, which is sequence of actions that transform the initial state, into a *final state* in which the goals hold. The plan is a solution of the planning problem. Usually, a state is represented by a finite collection of logical sentences. Actions are described by so-called *operators*. Following the STRIPS representation of Fikes and Nilsson (1971), operators are data structures that consist of preconditions and effects. A precondition is a conjunctive formula that must hold in a current state for the operator to be applicable. The effects describe how the state changes when the operator is applied. For more detailed introduction to planning, see e.g., (Russell and Norvig 1995).

1.2.1 The Logistics Transportation Domain

Let us illustrate these notions by an example from the logistics transportation domain (Veloso 1994). In this domain, there are different sorts of locations and means of transportation (see Figure 1.1). The initial state of a problem describes a certain configuration of objects, locations, and transportation means and a goal is to place certain objects at target locations. Figure 1.1 (a) shows an example of an initial state. There is a post office, *p4*, and an airport *a12*. There are two transportation means: a truck, *tr2*, located at *p4* and an airplane, *p15*, located at *a12*. Finally, there are two packages, *ob3* and *ob8*. The first one is loaded in *tr2* and the second one is located at *p4*. The final state consist of two goals: *ob3* must be located in *p12* and *ob8* must be loaded in *tr2*



(a) Initial State

****Operator:** *load-truck*
Parameters:
(<obj> OBJECT)
(<tr> TRUCK)
(<loc> LOCATION)
Preconditions:
at-obj(<obj> <loc>)
at-truck(<tr> <loc>)
Effects:
add: in-truck(<obj> <tr>)
del: at-obj(<obj> <loc>)

(b) An operator

Fig. 1.1. Initial state of a problem and an operator in the logistics transportation domain

Figure 1.1 (b) shows an example of an operator, *load-truck*, loading the object <*obj*> in <*tr*>. The preconditions state that <*obj*> and <*tr*> must be in the same location, <*loc*>. Effects are divided into *add* and *del* lists indicating literals added to and deleted from the state of the world, respectively. Figure 1.2 shows a plan achieving the two goals. Initially, as both *ob8* and *tr2* are located at *p4*, the preconditions of *load-truck* are met and this operator is applied. *tr2* is then moved to *a12*, where *ob3* is unloaded and loaded into *p12*. After performing this plan, both goals are met.

load-truck(ob8,tr2)—move-truck(tr2,p4,a12)—unload-truck(ob3,tr2)—load-plane(ob8,p15)

Fig. 1.2. A plan consisting of four actions

1.2.2 Why is Planning Difficult ?

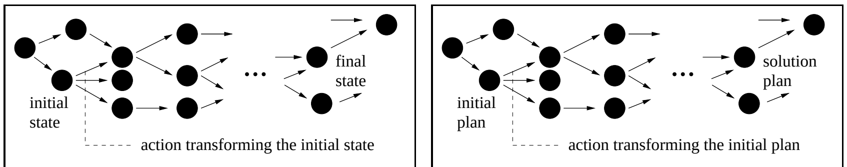
Although the previous example may seem simple, in fact, planning is a very difficult task. There may be several actions that can change the state of world. In the state illustrated in Figure 1.1 (a), dropping *ob3* in *p4* would have been a valid action, but it does not contribute to achieving the goals because an airplane cannot be moved to a post office. Typically, at any time of the planning process, there are several applicable operators but only few will lead to a solution. Even worse, it can happen that none of them might be useful. In this situation, a state will be achieved in which no action can be taken and backtracking is necessary to revise the actions taken previously. In terms of computational complexity theory, the problem of finding a correct plan for a given planning problem is an NP-complete problem (Bylander 1991). This results shows the intractability of the search involved in planning and calls for ways to restrict the search space.

1.2.3 Approaches to Planning

Several approaches to planning have been developed since the early days of STRIPS. Basically, all planners perform some kind of search; they differ, however, in the space they are searching and in the search strategy they use. The two dominant approaches search the *state space* and the *plan space*, respectively.

Figure 1.3 (a) illustrates the search space in state-space planning. The nodes represent the states of the world and the edges represent valid actions transforming states of the world. That is, actions transform states into states. From this point of view, the initial and final state are two nodes in the search space and a solution is a directed path from the initial to the final state.

In plan-space planning, operations transform partially ordered plans into partially ordered plans as illustrated in figure 1.3 (b). The *initial plan* is a



(a) Search in space of states (b) Search in space of plans

Fig. 1.3. Search in the space of states and in the state of plans

partially ordered plan representing the problem. This plan must be transformed into a *solution plan*. There are two types of operations transforming plans: operations achieving goals and operations solving conflicts. To achieve a goal, an action is introduced into the current partial plan. Actions are partially ordered according to whether an action achieves a precondition of the other action. Conflicts may occur due to the partial order between the plan steps which can be solved by introducing additional ordering constraints into the current plan. For example, in the logistics transportation domain different steps may require a truck to be available. A conflict occurs when there are not enough trucks available. To solve this conflict, some of the steps must be ordered to ensure that every truck is used one at the time. In so-called least-commitment planners an order between actions is introduced into the plan only when necessary. This is advantageous because additional ordering constraints can be introduced during planning.

Several studies have been made for indicating in which situations it is better to search in the space of plans rather than in the space of states and vice versa (Kambhampati et al. ; Veloso and Blythe 1994; Barrett and Weld 1994). Independent of the planning approach used, guidance is still needed to navigate through the exponential search space. We will now see, how case-based reasoning can provide this guidance.

1.3 Case-Based Planning

We now describe a general framework for case-based planning based on the CBR process model by Aamodt and Plaza (1994). This covers most case-based planning systems including those approaches developed at the CMU, at the Universities of Kaiserslautern, and at the University of Saarbrücken and Edinburgh.

1.3.1 Retrieval and Organization of the Case-Base

A case in case-based planning consists of a problem (initial and goals) and its plan. Given a new problem, the objective of the retrieval phase is to select a case from the case base whose problem description is most similar to the description of the new problem. Because of the usually needed adaptation in case-based planning, the retrieval should select adaptable cases (Smyth and Keane 1993). More precisely, the main goal of the similarity assessment is to predict the effort required for reusing the solution to solve the new problem. Therefore, a case should be considered very similar to the new problem, if only little effort is required for adapting its solution to the needs of the target and less similar if the adaptation is computationally expensive. In case-based planning, the reusability of a case is strongly determined by the solution contained in the case and not only by the problem description. Therefore, the similarity assessment refers to the fragments of the problem description which are relevant for successfully reusing the plan. One way to extract the fragments is to compute the weakest preconditions of the plan. This computation employs the domain knowledge about the available operators that ensures that the plan can be successfully applied. The similarity will then be assessed based on the fragment of the conditions that are satisfied in the current problem to be solved.

As in case-based reasoning in general, computing the similarity assessment may become very expensive when the case base has reached a considerable size. In this case, a trade-off between the objective to find the best case and the objective of minimizing the retrieval time exists as depicted in Figure 1.4. As the number of cases visited during retrieval increases, more time must be spent for retrieval (see curve 2) but better cases resulting in a shorter adaptation time will be found (see curve 1). Up to a certain point (optimal point), the total case-based planning time (retrieval+reuse, see curve 3) decreases when more cases are visited during retrieval. However, beyond this point the total planning time increases again if more cases are visited, because the possible gain through finding better cases does not outweigh the effort of finding them.

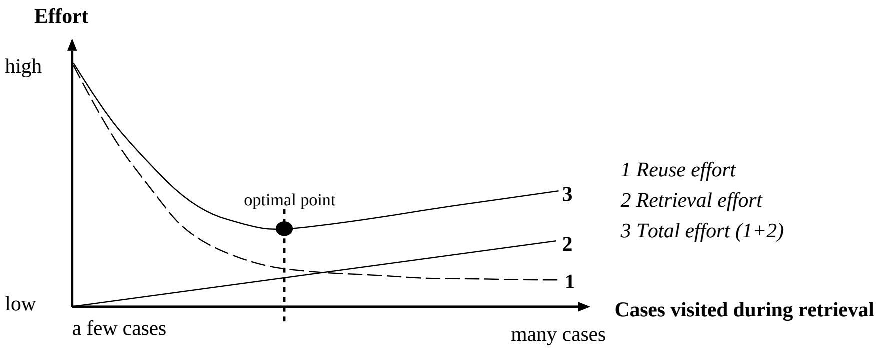


Fig. 1.4. Trade-off between retrieval effort and reuse effort (adapted from Veloso)

1.3.2 Reusing Previous Solutions

In case-based reasoning, at least two different kinds of approaches to reuse can be distinguished: *transformational adaptation* and *generative adaptation* (Carbonell 1983; Carbonell 1986). Transformational adaptation methods usually consist of a set of domain-dependent knowledge which directly modify the solution contained in the retrieved case, based on the difference between the problem descriptions in the case and of the current problem. While in early case-based planning systems (e.g., in Chef, Hammond 1986) only transformational adaptation was used, most recent systems use generative adaptation. For generative adaptation, the integration of a case-based problem solving strategy and a generative problem solver is central. The retrieved solution is not modified directly, but is used to *guide* the generative problem solver to find a solution. A basic principle is the replay of *decisions* that where made during the process of solving the problem recorded in the case. When replaying the solution trace of a previous case, some decisions can be reused, while the remaining decisions are taken by replaying another case or by using a generative planner. The result of this reuse phase is either a correct solution (w.r.t. the domain model) or the indication of a failure in case the problem could not be solved with allocated time resources. In different case-based planners, the particular guidance that a case provides depends on the type of generative problem solver that is used.

1.3.3 Revision of Solutions

The goal of the revision phase is to validate the computed solution in the real world or in a simulation of it. Due to the correctness of the reuse-phase, the resulting solutions are known to be correct w.r.t the domain model. Consequently, the simulation of the solution cannot contribute to an additional validation. Therefore, the solution must be validated in the real world. However, no methodological support of this kind is provided by today's case-based planning systems.

1.3.4 Retaining New Cases

In case-based planning, storing a new case in the case base requires far more effort than case-based reasoning for analytic tasks. While the latter simply stores a new case "as it is", case-based planning requires determining the "right" goals to index the cases. Roughly speaking, the goals used for indexing are those goals that are required for or affected by the taken decisions stored in the case. Furthermore, the decisions taken by the generative problem solver must be captured in stored cases.

1.4 PRODIGY/ANALOGY

PRODIGY/ANALOGY was the first system that achieved a complete synergy between generative planning and case-based planning (Veloso 1994) and used for the first time a full automation of the complete CBR-cycle. PRODIGY/ANALOGY has been developed within the PRODIGY planning and learning architecture (Carbonell et al. 1991). The generative planner is a means-ends analysis backward-chaining nonlinear planner, performing state-space search. The integration is based on the derivational analogy method (Carbonell 1986). This is a *reconstructive* method by which *lines of reasoning* are transferred and adapted to a new problem as opposed to transformational methods that adapt directly final solutions. PRODIGY/ANALOGY was and continues to be demonstrated in a variety of domains.

1.4.1 Retain: Generation of Planning Cases

A planning case to be stored consists of the successful solution trace augmented with justifications, i.e., the derivational trace. The base-level PRODIGY4.0 reasons about multiple goals and multiple alternative operators relevant to achieving the goals. This choice of operators amounts to multiple ways of trying to achieve the same goal. PRODIGY/ANALOGY provides a *language* to capture mainly three kinds of justifications for the decisions made during problem solving: links among choices capturing the goal dependencies, records of failed explored alternatives, and pointers to any external used guidance. We discovered in PRODIGY/ANALOGY that the key feature of this language is that it needs to be re-interpretable at planning replay time.

Automatic generation of the derivational planning episodes occurs by extending the base-level generative planner with the ability to examine its internal decision cycle, recording the justifications for each decision during its search process.

1.4.2 Indexing and Retrieval of Cases

From the exploration of the search space and by following the subgoaling links in the derivational trace of the plan generated (Carbonell 1986), the system identifies, for each goal, the set of *weakest preconditions* necessary to achieve that goal. The so called *foot-print* of a goal conjunct of the problem is recursively created by doing goal regression, i.e. projecting back its weakest preconditions into the literals in the initial state (Waldinger 1977). Goal regression acts as an explanation of the successful path. The literals in the initial state are therefore *categorized* according to the goal conjunct that employed them in its solution.

The system automatically identifies the sets of interacting goals of a plan by partially ordering the totally ordered solution found. The connected components of the partially ordered plan determine the independent fragments of the case each corresponding to a set of interacting goals. Each case is multiply indexed by these different sets of interacting goals.

When a new problem is presented to the system, the retrieval procedure must match the new initial state and the goal statement against the indices of the cases in the case library.

The retrieval algorithm focuses on retrieving past cases where the planner experienced equivalent goal interactions and has a reasonable match between initial states expecting therefore to achieve a large reduction in the new planning search effort.

1.4.3 Reuse: Replay of Multiple Planning Episodes

Prodigy/Analogy can construct a new solution from a set of guiding cases as opposed to a single past case. Complex problems may be solved by resolving minor interactions among simpler past cases.

Consider the logistics transportation domain. In this domain packages are to be moved among different places, by trucks and airplanes. The example below is simple for the sake of a clear illustration of the replay procedure. Extensive empirical results on prodigy/analogy have shown that the system scales well in problem complexity (Veloso 1994).

Figure 1.5 shows a new problem and two past cases selected for replay. The cases are partially instantiated to match the new situation. Further instantiations occur while replaying.

Past cases	*New problem*
(goal (inside-airplane ob3 pl5)) *(relevant-state* (at-obj ob3 <ap3>) (at-airplane pl5 a12))	*(goal* (inside-airplane ob3 pl5) (inside-truck ob8 tr2))
(goal (inside-truck ob8 tr2)) *(relevant-state* (at-obj ob8 p4) (at-truck tr2 <ap7>))	*(initial-state* (inside-truck ob3 tr2) (at-truck tr2 p4) (at-airplane pl5 a12) (at-obj ob8 p4))

Fig. 1.5. Instantiated past cases cover the new goal and partially match the new initial state. Some of the case variables are not bound by the match of the goals and state.

Figure 1.6 shows the replay episode to generate a solution to the new problem. The new situation is shown at the right side of the figure and the two past guiding cases at the left.

The transfer occurs by interleaving the two guiding cases, performing any additional work needed to accomplish remaining subgoals, and skipping past work that does not need to be done. In particular, the case nodes *cn3'* through *cn5'* are not reused, as there is a truck already at the post office in the new problem. The nodes *n9-14* correspond to unguided additional planning

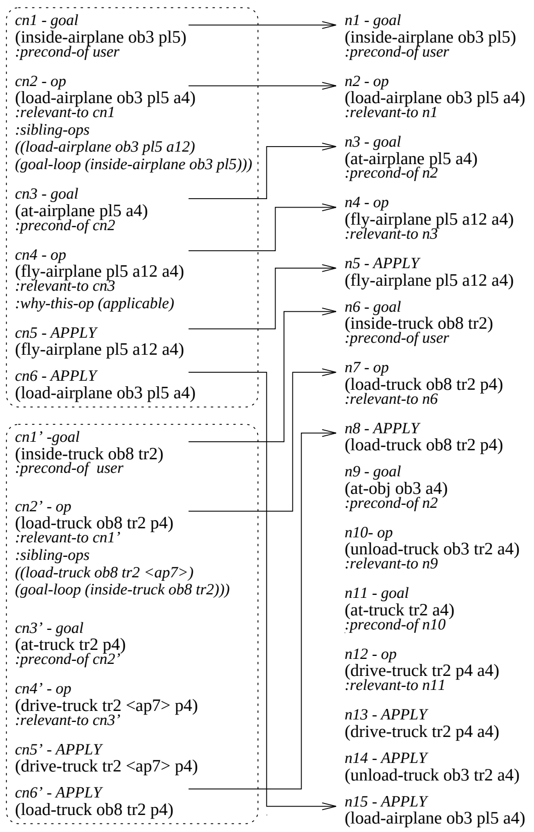


Fig. 1.6. Derivational replay of multiple cases

done in the new episode.[1] At node *n7*, Prodigy/Analogy prunes out an alternative operator, namely to load the truck at any airport, because of the recorded past failure at the guiding node *cn2'*. The recorded reason for that failure, namely a goal-loop with the *(inside-truck ob8 tr2)*, is validated in the new situation, as that goal is in the current set of open goals, at node *n6*. Note that the two cases are merged using a bias to postpone additional planning needed. Different merges are possible.

1.5 CAPlan/CbC : Plan Reuse in the Space of Plans

CAPlan/CbC (Muñoz-Avila et al.) is a generic case-based reasoning system built on top of CAPlan (Weberskirch 1995), a plan-space planner (McAllester and Rosenblitt). CAPlan/CbC is motivated in developing techniques for case-based planning in complex domains such as the domain of process planning for manufacturing workpieces. CAPlan/CbC focuses on developing techniques for retrieval in technical domains and for reuse in plan-space planners.

1.5.1 Process Planning for Manufacturing Mechanical Workpieces

At the beginning of the manufacturing process a piece of raw material and the description of a workpiece are given. Descriptions of the cutting tools and clamping material available are given as well. The problem is to remove layers of raw material in order to obtain the workpiece. Typically, the piece of raw material is clamped on a lathe machine that rotates at a very high speed. A cutting tool is used to remove layers of raw material. The process continues by replacing the cutting tool or changing the clamping position to process areas of the workpiece. Figure 1.7 shows an intermediate stage during this process. The grid area corresponds to the portion of raw material that still needs to be removed. The parts presented there correspond to the two sides of a workpiece, two ascending outlines and a so-called undercut.

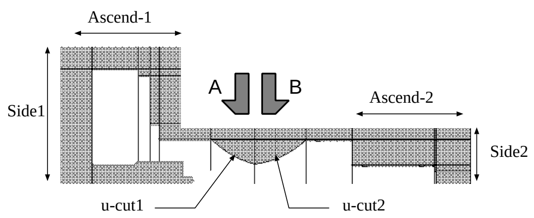


Fig. 1.7. Half display of a workpiece and two cutting tools.

[1] Note that extra steps may be inserted at any point, interrupting and interleaving the past cases, and not just at the end of the cases.

Each goal of a problem corresponds to machining an area of a given workpiece. The initial state is a symbolic description of the geometry of the processing areas, the cutting tools and the clamping material available. There are different types of areas. For each type, there is at least one operator indicating the conditions that must be met to machine it. Typically, a clamping position on the workpiece must be determined. Also, according to the clamping position and the type of area a certain kind of cutting tool must be held. Other operators represent different clamping operations and regulate the use of the cutting tools. A basic restriction regarding the clamping and holding operations is that at any time of the machining process the workpiece is clamped from at most one position and a limited number of cutting tools is being held.

1.5.2 Dependency-Driven Case Retrieval

In the domain of process planning, it is possible to *predetermine* restrictions on the order for manufacturing certain parts of the workpiece. These ordering restrictions are determined based on the geometry of the workpiece and are independent of the cutting tools available. For example, at the beginning of the manufacturing process of the workpiece depicted in Figure 1.7, the undercut was covered by the area *intermediate* (not shown in Figure 1.7). Clearly, the area *intermediate* must be processed before the undercut. Thus, for any plan manufacturing this workpiece, the goal corresponding to processing *intermediate* is achieved before the goal achieving the undercut. In CAPLAN/CBC these ordering restrictions are computed by a geometrical reasoner (Muñoz-Avila and Hüllen ; Muñoz-Avila and Weberskirch). In a more general context there are three possible sources for these ordering restrictions, which we denote by $\prec$:

- *A domain specific reasoner.* A typical example is the geometrical reasoner.
- *The user.* The user may specify additional restrictions because of quality considerations that are not explicitly represented in the domain theory.
- *Static analysis.* In principle, the domain theory can be analyzed to predetermine some ordering restrictions (Etzioni 1993).

We call dependencies to the ordering restrictions between the goals of a solution plan motivated in the domain of process planning, were the way a processing area is manufactured depends on how previous processing areas are manufactured. The initial situation, the goals, and the ordering restrictions form *extended problem descriptions* (Muñoz-Avila and Hüllen ; Muñoz-Avila and Weberskirch). For a case to be retrieved, the dependencies in the case (i.e., the ordering in which the goals are achieved), denoted by $\rightarrow$, must be consistent with the ordering restrictions of the problem. That is, if two goals in the new problem g_1, g_2 have the ordering restriction $g_1 \prec g_2$, then, for the corresponding goals in the source case, g'_1, g'_2, the relation $g'_2 \rightarrow g'_1$

must *not* hold. An architecture of the case base has been developed that allows to find cases whose dependencies are consistent with the ordering restrictions of new problem in an efficient way. The dependencies are the main criterion to discriminate cases in the case base. We called this retrieval strategy *dependency-driven* as a clear contrast to other case-based planners where cases are retrieved based on the goals that they achieved (i.e., goal-driven). The dependency-driven retrieval strategy is adequate for domains in which a partial order between the goals can be predetermined. See (Muñoz-Avila and Hüllen) for further details.

1.5.3 Adaptation with Complete Decision Replay

The adaptation method conceived and implemented in CAPlan/CbC is called complete decision replay. This method is fundamented on the way the base-level planner CAPlan is built (Weberskirch 1995). To represent knowledge about plans and contingencies that occur during planning, CAPlan is built on the generic REDUX architecture (Petrie 1991). The REDUX architecture represents relations between goals and operators and between operators and subgoals. In the parlance of REDUX a *decision* is made when an operator is applied to achieve a goal. Decisions are represented as a subtree in the *goal graph*. It represents basic dependencies between goals and subgoals as well as between subgoals and decisions. A key aspect in CAPlan is that the *justifications* of all decisions (valid and not valid) are always maintained. A justification has the form $\{a_1, a_2, ..., a_n\}$, where a_i is a ordering or a variable constraint. For example, consider a decision corresponding to the application of an operator that requires $X = Y$ to be true, where X and Y are variables. If the decision is valid, an example of a possible justification is $\{X = 1, Y = 1\}$. An example of a justification if the decision is invalid is $\{X = 1, Y = 0\}$.

The complete decision replay method consists of reconstructing the goal graph relative to the new situation. The validity of a decision (i.e., if the decision failed or not) is stated if it's justifications can be reconstructed relative to the new situation. The partial solution represented in the reconstructed goal graph is then completed by first principles planning. Three major advantages can be observed as a result of this process:

1. The user may interact during the completion process. The user may prune parts of the replayed case or neglect the validity of certain problem conditions. Because CAPlan/CbC reconstructs the goal graph, the functionality of the base-level planner CAPlan is preserved. Thus, a plan can be generated by modifying the current plan without having to plan from the scratch.
2. Powerful backtracking methods can be used. CAPlan performs dependency-directed backtracking based on the goal graph reconstructed by CAPlan/CbC.

3. CAPLAN does not explore failed attempts. Having found that the justifications for a failed decision can be reconstructed in the new situation CAPLAN/CBC prunes completion possibilities that otherwise could have been explored during the completion process.

The principle of considering the failed attempts during reuse is illustrated by continuing with the example depicted in Figure 1.7. This figure depicts an intermediate stage during this process in which the undercut has not been processed. Undercuts always can be decomposed in two parts (labeled *u-cut1* and *u-cut2*). In this figure a left and a right cutting tools are also shown, labeled *A* and *B* respectively. For manufacturing the undercut the workpiece needs to be clamped from an ascending outline, for example *Ascend-1*. The left tool is used for removing the left part (i.e., *u-cut1*). For removing the right part (i.e., *u-cut2*) there are three possibilities: (1) to use the right tool, (2) to clamp the workpiece from the outline *Ascend-2* and use the same left tool again or (3) to clamp the workpiece from *Side2* and use the left tool. The last possibility requires that there is a perforation on *Side2*. Since in the example there is only a perforation on *Side1*, it will be discarded if considered by the planner.

The left side of Figure 1.8 outlines a plan for manufacturing the workpiece shown in Figure 1.7, under the supposition that there is a left and a right cutting-tools available. Dashed boxes represent plan-steps and the arcs pointing downwards indicate the partial-order for performing them. This plan states that for removing *u-cut1*, the workpiece must be clamp from *Ascend-1* and the left tool must be used. After that, *u-cut2* is removed by using the right tool. This plan contains also the information that clamping from *Side2* failed because it does not have any perforation (node labeled *R*).

Suppose that a new problem is given that consists of the same workpiece, but this time there is only a left tool available. For solving this problem the plan obtained with the two tools will be reused, as illustrated in Figure 1.8. The horizontal arrows show the decisions of the case that are replayed in the new situation. Particularly the decisions concerning the manufacturing of *u-cut1* can be replayed in the new situation. However, the decision concerning the manufacturing of *u-cut2* cannot be replayed, since in the new situation there is no right tool available. As a result, a rejected decision is created (node labeled *S*). The rejection of the operator *clamping from Side-2* is replayed (node labeled *R'*), as in the new situation *Side-2* has no perforation.

Once the replay of cases is finished, the remaining goals need to be solved by the generative planner. As stated before, the key issue is the generative planner (CAPLAN) will avoid performing unnecessary backtracking. Particularly, for solving the goal corresponding to manufacturing *u-cut2*, CAPLAN will not pursue to use the right tool, neither to clamp from *Side2*. Instead, it will select to clamp the workpiece from *Ascend-2* (arc labeled *P*), which is the right choice.

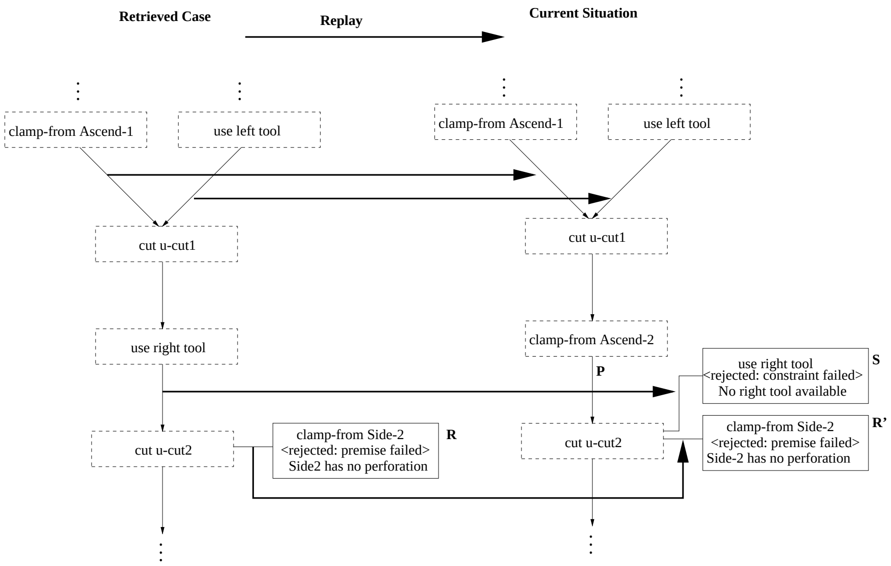


Fig. 1.8. Example of replay in CAPlan/CbC.

1.6 Summary of Theoretical and Experimental Results

CAPlan/CbC has been evaluated in the domain of process planning as defined in Muñoz-Avila and Weberskirch (1996). This specification consists of 33 types of objects and 27 operators. We formally prove in Muñoz-Avila and Weberskirch (1996) that this specification meets the conditions stated in Kambhampati et al. () and Barrett and Weld (1994). These conditions state situations in which using a plan-space planner is better than a state-space planner. As a result we conclude that planning with this specification is more adequate with a plan-space planner like CAPlan. Given that the first-principles planner plays an important role with reuse by complete decision replay, this result supports the use of a case base planning in CAPlan.

Experiments suggest that dependency-driven retrieval increase the accuracy of the retrieval (Muñoz-Avila and Hüllen) and reduce the effort of the first-principles planner to complete the partial solution obtained after replay. As a result, the performance of the overall case-based planning process is increased (Muñoz-Avila and Weberskirch).

Finally, the experiments performed show that the time taken by the first-principles planner to complete the partial solution obtained after replay is reduced when performing complete decision replay (Muñoz-Avila and Weberskirch).

1.7 PARIS: Flexible Reuse of Cases at Different Levels of Abstraction

Traditionally, case-based reasoning approaches retrieve, reuse, and retain cases given in a single, concrete representation. PARIS[2] (Bergmann 1996; Bergmann and Wilke 1995a) is a domain independent case-based planning system that differs from this traditional approach in that it introduces abstraction techniques into the case-based reasoning process. PARIS retrieves, reuses and retains cases at different (higher) levels of abstraction.

In a nutshell, PARIS works as follows. Available planning cases given at the concrete level are abstracted to several levels of abstraction which leads to a set of *abstract cases* that are stored in the case-base. Case abstraction is done automatically in the retain phase of the CBR-cycle. When a new problem must be solved, an abstract case is retrieved whose abstract problem description matches the current problem at an abstract level. In the subsequent reuse phase, the abstract solution is refined, i.e., the details that are not contained in the abstract case are added to achieve a complete solution of the problem. This refinement is done by a generative planner that performs a forward directed state space search. Figure 1.9 shows an overview of the whole system and its components. Besides case abstraction and refinement, PARIS also includes an explanation-based approach for *generalizing cases* during learning and for *specializing* them during problem solving. This technique allows to further increase the flexibility of reuse.

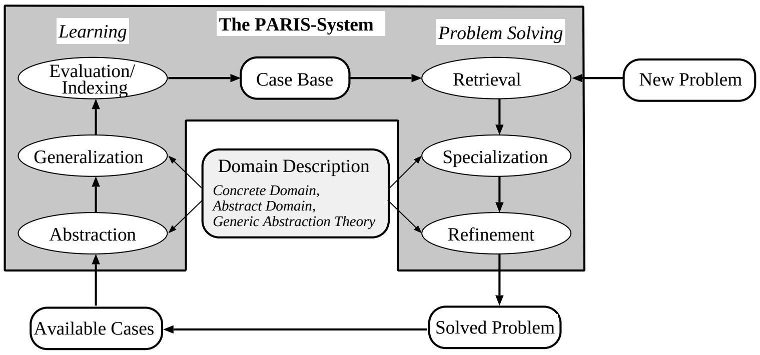


Fig. 1.9. Architecture of the PARIS system

The PARIS system benefits from using abstract cases in several ways (Bergmann and Wilke 1996):

– Abstraction reduces the complexity of a case, i.e., it simplifies its representation, e.g. by reducing the number of predicates, relations, constraints, operators, etc.

[2] Plan Abstraction and Refinement in an Integrated System.

- Cases at higher levels of abstraction can be used as a kind of prototype, which can be used as indexes to a larger set of related, more detailed cases.
- Cases at higher levels of abstraction can be used as a substitute for a set of concrete cases, thereby reducing the size of the case base.
- Abstraction increases the flexibility of reuse. Adapting abstract solutions contained in cases at higher levels of abstraction leads to abstract solutions suitable for a large spectrum of concrete problems.
- Abstraction and refinement, on their own, can be used as a method for solution adaptation.

These advantages are particularly valuable in case-based planning, since usually a large number of complex cases must be considered, the similarity assessment is very expensive, and since flexible adaptation is required because of the vast solution space.

We now explain the approach followed in Paris in more detail.

1.7.1 Different Levels of Abstraction and Abstract Cases

While cases are usually represented and reused on a single level, abstraction techniques enable a CBR system to reason with cases at several levels of abstraction. Firstly, this requires the introduction of several distinct levels of abstraction. Each level of abstraction allows the representation of problems, solutions, and cases as well as the representation of general knowledge that might be required in addition to the cases. Usually, levels of abstraction are ordered (totally or partially) through an abstraction-relation, i.e., one level is called *more abstract* than another level.

A more abstract level is characterized through a reduced level of detail in the representation, i.e., it usually consists of less features, relations, constraints, operators, etc. Moreover, abstract levels model the world in a less precise way, but still capture certain important properties.

In traditional hierarchical problem solving (e.g., Abstrips (Sacerdoti 1974)), abstraction levels are constructed by simply dropping certain features of the more concrete representation levels. However, it has been shown that this view of abstraction is too restrictive and representation dependent to make full use of the abstraction idea (Bergmann and Wilke 1995b; Holte et al. 1995). Therefore, Paris enables different levels of abstraction to have different representation languages. That is, abstract properties can be expressed in completely different terms than concrete properties which enables the representation of meaningful abstractions in planning domains. Therefore, Paris assumes that in addition to the concrete planning domain, an *abstract planning domain* composed of *abstract operators* is part of the general knowledge. The general knowledge also contains a set of *abstraction rules* that describe different ways of abstracting concrete states. For example in the domain of planning rotary symmetric workpieces, the concrete domain contains operators and predicates to describe the detailed contour of workpieces and individual cut operations that must be performed. The abstract domain abstracts from the detailed contour and represents larger units, called complex processing areas, together with the status of their processing (see also Figure 1.10).

Based on the level of abstraction, we can distinguish between two kinds of cases: *concrete cases* and *abstract cases*. A *concrete case* is a case located at the lowest available level of abstraction. An *abstract case* is a case represented at a higher level of abstraction.

1.7.2 Case Abstraction

Case abstraction means reducing the level of detail contained in the problem description and in the solution of a case, i.e., an abstract case contains less operators and less states than the concrete case. Furthermore, abstract operators and states are described using more abstract terms which typically require a reduced number of predicates.

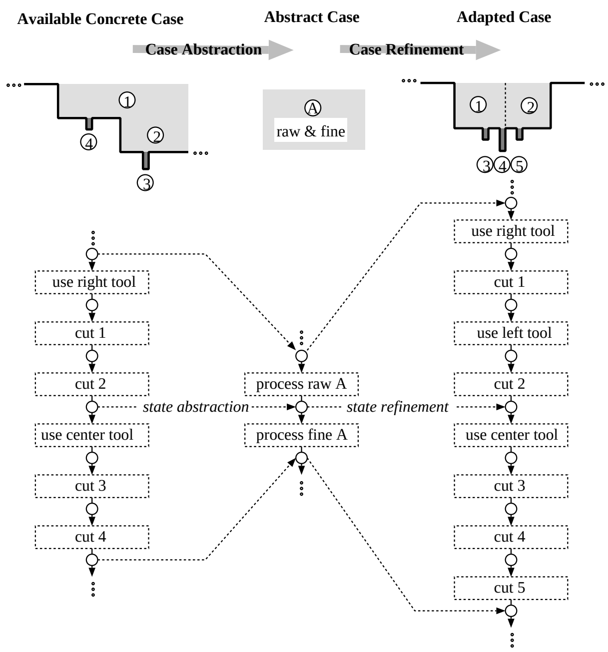


Fig. 1.10. Example of generating and refining abstract cases.

Figure 1.10 presents an example of the relationship between a concrete case and an abstract case. The left side shows a section of a concrete case, depicting how a step-like contour with two grooves is manufactured by a sub-plan consisting of 6 steps. The abstract case, shown in the middle of this figure, abstracts from the detailed contour and just represents a complex processing area named A that includes raw and fine elements. The corresponding abstract plan contains 2 abstract steps: processing in a raw manner and processing in a fine manner. The arrows between the concrete and the abstract case show how concrete and abstract states correspond. Each abstract state is derived from one of the existing concrete states (state abstraction). However, not all concrete states are abstracted. Some concrete states are skipped because they are considered an irrelevant detail. As a byproduct of this state abstraction, a sequence of concrete operators is abstracted to a single abstract operator. Note that the above explained kind of case abstraction is performed by an automatic procedure in Paris as part of the retain-phase of the CBR-cycle (see Bergmann and Wilke (1995a); Bergmann and Wilke (1995b) for details). Abstract and concrete cases are then stored in the case base for reuse.

1.7.3 Reuse of Abstract Cases

When a new problem must be solved, an abstract (or concrete) case is selected which matches the current problem at the respective level. If several matching cases are contained in the case-base, the retrieval procedure selects the case that is located at the lowest level of abstraction. The motivation for this preference is that for more concrete cases, less effort has to be spent during refinement to achieve a complete solution. This corresponds to a similarity measure that ranks two cases more similar, the lower the level of abstraction is, on which the problems are identical (Bergmann et al. 1993).

During the reuse phase, the abstract solution contained in the retrieved case must be refined to become a fully detailed complete solution. The right side of Figure 1.10 shows an example of such a refinement. While the contour of the two workpieces differs drastically at the concrete level, the abstract problem matches exactly because the 5 atomic contour elements in the new problem can be abstracted to a complex processing area with raw and fine elements. The abstract operators of the abstract case are then used to guide the state space planner to find a refined solution to the problem. Each abstract state is used as a sub-goal in the planning process. In the portion of the case shown in Figure 1.10, the abstract operator *process raw A* is refined to a sequence of four concrete steps which manufacture area 1 and 2. The next abstract operator is refined to a four-step sequence which manufactures the grooves 3, 4, and 5.

Note that Paris allows the *reuse of problem decompositions* at different levels of abstraction. Abstract plans decompose the original problem into a set of much smaller subproblems. These subproblems are solved by a search-based problem solver. The problem decomposition leads to a significant reduction of the overall search that must be performed to solve the problem. With pure search the worst-case time complexity for finding the required solution by search is $O(b^n)$, where n is the length of the solution and b is the average branching factor. If the problem is decomposed by an abstract solution into k subproblems, each of which requires a solution of length $n_1, \ldots, n_k$, respectively, with $n_1 + n_2 + \cdots + n_k = n$, the worst-case time complexity for finding the complete solution is $O(b^{n_1} + b^{n_2} + \cdots + b^{n_k})$ which is $O(b^{max(n_1, n_2, \ldots, n_k)})$. So, if the sub-problems are small enough and mostly independent from each other, the underlying planner is able to solve them without stepping into the intractability problem.

1.7.4 Organization of the Case Base

As introduced in Section 1.3.1, the organization of the case base plays an important role in case-based planning. In PARIS abstract cases located at different levels of abstraction are used as hierarchical indexes to those concrete (or abstract) cases that contain the same kind of information but at a more detailed level. For this purpose, an *abstraction hierarchy* is constructed during the retain phase, in which abstract cases at higher levels of abstraction are located above abstract cases at lower levels. The leaf nodes of this hierarchy contain concrete cases (see Fig. 1.11). During retrieval, this hierarchy is traversed top-down, following only those branches in which abstract cases are sufficiently similar to the current problem. This kind of memory organization is similar to the *memory organization packets* (MOPs) of Schank (1982).

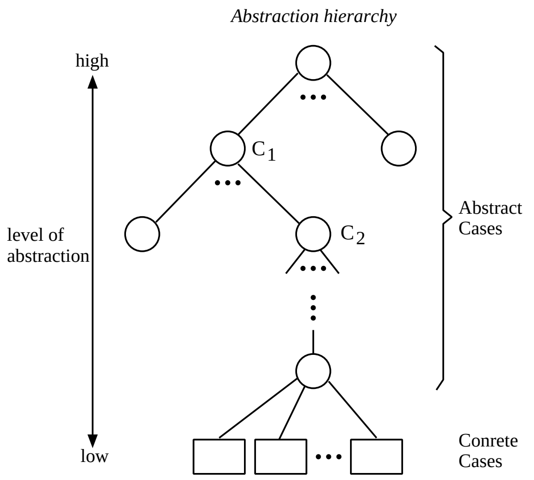


Fig. 1.11. Abstraction hierarchy for indexing cases.

An important advantage of such an abstraction hierarchy is that it provides a frame for realizing case deletion policies (Smyth and Keane). Cases deletion is particularly important to avoid the utility problem[3] (Tambe and

[3] The utility problem in CBR is also called swamping problem by Francis and Ram.

Newell 1988; Francis and Ram 1993) that occurs when case bases grow very large. When reusing abstract cases for indexing and reuse, case deletion can be efficiently realized through a pruning of the abstraction hierarchy, i.e. deleting some branches of the tree. If a certain branch of the tree is removed (together with the respective concrete and possibly abstract cases) the abstract cases that remain accessible can still cover the set of target problems previously covered by the deleted case. However, not all details are present any more. During reuse they must therefore be reconstructed by the generative planner. Consequently, pruning of the abstraction hierarchy has two contrary effects on the overall problem solving time:

- Since the detailed parts of the solution are not available any more, the *reuse effort increases* because these details must be reconstructed by the planner.
- Since the number of cases that must be inspected during retrieval is reduced, the *retrieval effort is reduced.*

The Paris system makes use of an elaborated cost model for determining the expected cost or benefit (retrieval effort + reuse effort) of removing certain parts of the abstraction hierarchy (and the case base) (Bergmann 1996). Based on this model, an optimization algorithm computes a pruned abstraction hierarchy and thereby the related fragment of the case base that leads to the lowest expected overall cost for solving a new planning problem.

1.7.5 Summary of Experimental Results

The Paris system was evaluated in extensive empirical studies using the domain of manufacturing planning. A detailed description of this domain can be found in Bergmann and Wilke (1995a); empirical results are presented in (Bergmann 1996; Bergmann and Wilke 1995b; Bergmann and Wilke 1995a; Bergmann and Wilke 1996). The most important results of these studies can be summarized as follows. For the used domain representation and case base it could be shown that:

- The case-based planning approach followed in Paris leads to significantly shorter problem solving time than a pure generative planner.
- The reuse of abstract cases leads to a significant increase in the flexibility of reuse (number of new problems for which a case can be efficiently reused).
- Organizing the case-base using an abstraction hierarchy leads to a significant reduction of the retrieval time compared to a linear retrieval approach.
- The case deletion policy (pruning of the abstraction hierarchy) leads to a significant reduction of the overall problem solving time.

1.8 ABALONE: Analogy in Proof Planning

Proof planning is an alternative to traditional methods in theorem proving that employs operators that represent chunks of a proof rather than low-level inference steps such as resolution. Proof planning is AI-planning in a rather complicated domain in which goals and state descriptions consist of complicated first-order or even higher-order sequents (axioms, lemmata, theorems, proof conjectures) rather than literals. In this domain, the enormous search spaces can be handled only by user-interaction or by very elaborate control knowledge. Often control knowledge that avoids extensive search is not available, and therefore, case-based planning is a possible strategy to overcome the control problems.

The proof planner CL^AM (Bundy et al. 1991) on top of which the analogy procedure ABALONE is implemented, has successfully been applied to theorem proving by induction. As known from Peano induction for natural numbers, inductive proofs have base-case and step-case subproofs. The latter has the subgoal $(IH \rightarrow IC)$ for an induction hypothesis IH and an induction conclusion IC and this subproof aims at rewriting the induction conclusion until the induction hypothesis is applicable.

For instance, in planning the theorem `lenapp`: $\forall a, b.\ len(app(a,b)) = len(app(b,a))$,[4] the operator INDUCTION computes the induction schema (here, induction on lists[5]) and outputs the base-case and the step-case subgoal $(IH \vdash IC)$ for the induction hypothesis IH

$$len(app(a,b)) = len(app(b,a)) \tag{1.1}$$

and the induction conclusion IC

$$len(app(\underline{::(h,}a),b)) = len(app(b,\underline{::(h,}a))) \tag{1.2}$$

which is depicted as a tree in Figure 1.12(A). (The underlines mark the differences between IC and IH (*contexts*) that are shown as circles in the figure.) The operator WAVE applies rewrite-rules to subgoals, and three applications of WAVE reduce the goal $IH \vdash IC$ to

$$IH \vdash \underline{s}(len(app(a,b))) = \underline{s}(len(app(b,a))). \tag{1.3}$$

This subgoal is represented by a tree in Figure 1.12(B). The operator FERTILIZE uses the induction hypothesis to rewrite the current goal. Hence, FERTILIZE reduces the subgoal (1.3) to the subgoal

$$H \vdash \underline{s}(len(app(b,a))) = \underline{s}(len(app(b,a))) \tag{1.4}$$

[4] $app, len, ::$ denote the list functions $append, length, cons$. s is the successor function on natural numbers.

[5] with the successor $::(h,x)$ of x,

which can be removed by the operator ELEMENTARY. The whole step-case proof plan of `lenapp` is depicted in the lhs of Figure 1.13.

The control in *CIAM* bases on the *rippling* heuristic that guides the rewriting of the induction conclusion in a systematic way that reduces the differences (context) between IC and IH. From an abstract point of view, rippling removes contexts or moves them to the top of the theorem tree (the term tree representation of a theorem) as shown in Figure 1.12, where the bold paths are those on which the contexts are moved. As soon as the contexts sit in the top position, FERTILIZE can be applied. Rippling is a powerful search

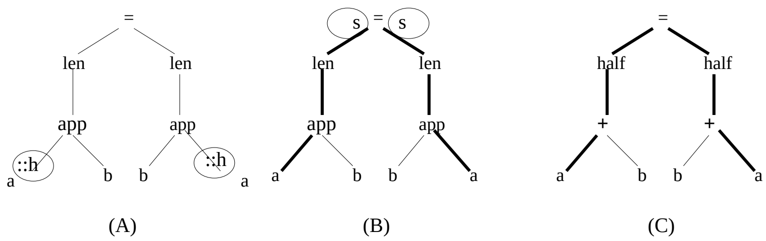


Fig. 1.12. Term trees (A) of the induction hypothesis of `lenapp`, (B) of the solved `lenapp`, and (C) of `halfplus` – with rippling paths in bold and ovals for contexts (omitted in C)

heuristic, but even with such a heuristic fully automated proof planning can fail because the control heuristics can be too restrictive, the default set of operators can be too restrictive, or the search for lemmata that are missing in the target becomes intractable. Using analogy as a control strategy in proof planning can help in these situations, see (Melis and Whittle 1997). It can replay existing proof plans with the purpose to

- suggest operators rather than searching for them,
- override the default control and default configurations,
- replace search-intensive subtasks (such as finding an induction schema),
- suggest target lemmata, and to
- avoid user interaction.

1.8.1 Analogy-Driven Proof Plan Construction

According to the model of analogy-driven proof plan construction in (Melis a) that is supported by empirical observations on human theorem proving (Melis 1994), ABALONE works at the proof plan level. Analogically replaying high-level plans is more robust than analogically replaying, e.g., resolution steps because, while for many problems the exact replay of resolution steps fails, the replay of higher-level proof plan operators may succeed. ABALONE analogically transfers a source plan produced by *CIAM* to a plan for the target problem. The transfer is done on the basis of second-order mappings, presented in the following subsection, that can map function and relation symbols. The replay of the source plan is made node by node. As in PRODIGY/ANALOGY, at each node in the source plan justifications are stored and during the replay, these justifications are reinterpreted as described below. Of course, the actual justifications are specific for theorem proving, e.g., capturing the existence of a lemma needed in the proof, the identity of function occurrences in a formula. A source node is only replayed if its justifications still hold in the target or if the justification can be established. This serves two purposes. First, it guarantees the correctness of target operator applications. Second, it provides a way of transferring only part of the source plan in some cases: if justifications cannot be established in the target, a *gap* is left in the target plan. In this way, it is only this part of the source that is not transferred, whereas the replay of the rest of the source plan can still be attempted.

Sometimes a node by node replay as described above is insufficient because the source and target plans differ in a significant respect. An example might be if the source and target plans have different induction schemes or operators have to be duplicated or replaced. Rather than failing, ABALONE is equipped with a number of *reformulations* introduced in (Melis a) and (Melis and Whittle). which make additional changes in the target plan. These reformulations are triggered by peculiarities of the mappings and justifications and are applied before the replay. Figure 1.13 shows an example of an analogical replay. The equations displayed in the source plan are justifications explained in section 1.8.3.

The main steps in the analogy-driven proof plan construction (Melis a) can be summarized as:

- Retrieve a source problem.
- Find a second-order mapping m_b from the source theorem to the target theorem.
- Extend m_b to a mapping m_e from source rules to target rules.
- Decide about the reformulations to be applied. The need for a reformulation is triggered by patterns in m_b or m_e.
- Following the source plan, analogically replay the operators by
  - Apply reformulations
  - **if** operator justifications hold, **then** apply operator in target, **else** try to establish justification.
  - **if** justification cannnot be established, **then** leave gap in target plan.

Second-order mappings and matching are those that can send function symbols to function terms. Pure object-level syntactic similarity measures do not work for the retrieval and the mapping of proof plans because small changes of the problems can cause tremendous changes in proof plans. Therefore, ABALONE employs proof-relevant LF-abstractions, explained below , to restrict the retrieval, mapping, and adaptation . The restrictions derived from the abstractions guarantee the replayed (step-case) plan to be a correct plan for the target problem.

SOURCE PLAN

$$len(app(::_5\ (h, a), b)) = len(app(b, ::_5\ (h, a)))$$

$::_5 = ::_1$ $\quad\downarrow$ WAVE (app2)

$$len(::_2\ (h, app(a, b))) = len(app(b, ::_5\ (h, a)))$$

$::_2 = ::_3$ $\quad\downarrow$ WAVE (len2)

$$s_1(len(app(a, b))) = len(app(b, ::_5\ (h, a)))$$

$::_5 = ::_4$ $\quad\downarrow$ WAVE (lenapp2)

$$s_1(len(app(a, b))) = s_2(len(app(b, a)))$$

$\downarrow$ FERTILIZE

$$s_1(len(app(b, a))) = s_2(len(app(b, a)))$$

$s_1 = s_2$ $\quad\downarrow$ ELEMENTARY

true

TARGET PLAN

$$half(+(s(s(a)), b)) = half(+(b, s(s(a))))$$

$\downarrow$ WAVE (plus2)

$$half(s(+(s(a), b))) = half(+(b, s(s(a))))$$

$\downarrow$ WAVE (plus2)

$$half(s(s(+(a, b)))) = half(+(b, s(s(a))))$$

$\downarrow$ WAVE (half3)

$$s(half(+(a, b))) = half(+(b, s(s(a))))$$

$\downarrow$ WAVE (lemma)

$$s(half(+(a, b))) = s(half(+(b, a)))$$

$\downarrow$ FERTILIZE

$$s(half(+(b, a))) = s(half(+(b, a)))$$

$\downarrow$ ELEMENTARY

true

Fig. 1.13. Step-case replay

1.8.2 Retrieving Adaptable Cases

In the retrieval, second-order mappings are used to map the source theorem to the target theorem and source rules to target rules. First, a *constrained* basic mapping m_b is constructed that maps the source theorem with indexed function symbols to the given target theorem. m_b is then augmented by an extended mapping m_e which maps the source to the target rewrite-rules. m_b is restricted to essentially preserve the abstractions explained next. This constraint guarantees a successful step-case target plan.

Labeled fragments (LFs), introduced in Hutter (1994), are an abstraction of annotated rewrite-rules obtained by removing the structure of the contexts and those parts of the rules not affected by contexts. For each function/relation symbol occurring in the source and target theorem LFs corresponding to the rewrite-rules that belong to the source and target problem, respectively, are automatically computed. Take, for instance, the rewrite-rule `app2`: $app(\underline{::(X,Y)}, Z) \Rightarrow \underline{::(X,}(app(Y,Z)))$. Note that in the lhs of `app2` the context is situated at the left argument of *app* and how it moves to the top of *app* in the rhs of `app2`. This situation is reflected in the most right labeled fragment of *app* as shown in Figure 1.14.

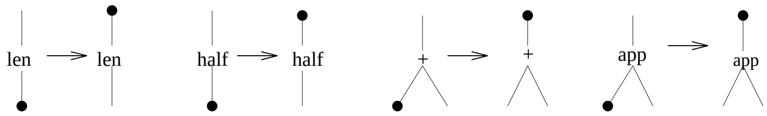


Fig. 1.14. Labeled fragments

The rippling paths in a theorem tree abstractly encode the step-case proof, in particular, the consecutive application of the WAVE operator in proof plans. In turn, the LFs determine the rippling paths. Therefore, LFs provide a **proof-relevant abstraction** of problems to be proved by induction. In the example, the WAVE operators (Figure 1.13) apply rewrite-rules, such as `app2`, that move the context as abstractly shown in Figure 1.14. First WAVE(app2) is applied and then WAVE(len2) which corresponds to the (abstract) rippling path of `lenapp` in Figure 1.12(B) which, in turn, is determined by the LFs in Figure 1.14. Based on corresponding LFs (Figure 1.14), the rippling path of `lenapp` equal those of `halfplus` in Figure 1.12(C) and hence the proof plan will be very similar.

With the help of LFs, the case base can be pre-structured into classes of cases. The elements of a class can be further distinguished by their rewrite-rules. This makes the retrieval a **two-stage** process with a first cheap step that retrieves the source class and a second, more expensive step that tries to second-order map the rewrite-rules of each case of that class rewrite-rules of the target problem. A class of source cases contains all cases with identical rippling paths. A class is represented by a theorem tree whose nodes on rippling paths are annotated with the common LFs. The nodes occurring outside of rippling paths in any of the cases are abstracted to meta-variables in the class representation because they are irrelevant for the abstraction. The first stage tries to approximately match the target theorem with one of the class representatives and checks the LFs in this process as well. It chooses the class with the fewest differences in rippling paths that are predefined by the available reformulations.

The second stage chooses the best case from the retrieved class by trying to second-order map the target-rules (lemmata) with the rules of the cases in the retrieved class. Second-order mapping is decidable but can be expensive, in particular with many rules involved. The heuristics in Whittle (1995) support choosing reasonable mappings.

1.8.3 Analogical Replay

During the planning of the source theorem, justifications, i.e. reasons for the application of an operator, are stored at each source plan node. As opposed to PRODIGY/ANALOGY, ABALONE's justifications consist of certain operator preconditions and of *C-equations*: Such a precondition of the WAVE operator is the existence of a rewrite-rule that matches the current goal. *C(onstraint)-equations* are justifications that are due to the use of indexed functions. ABALONE is able to map a function symbol at different positions in the source to different target images. For this reason, source function symbols at different positions are differentiated by indices. During the source planning, constraints of the form $f_i = f_j$, called C-equations, may be placed on these functions. In Figure 1.13,[6] for instance, $::_2 = ::_3$ states that the function $::_2$ in the subgoal is equal to the function $::_3$ in the rewrite-rule `len2` employed by WAVE in the source. In the target this justification requires the identity of the images of mapping $::_2$ and $::_3$ to target functions. Note how the C-equations in Figure 1.13 arise in the source: When WAVE(`app2`) is applied, the lhs of `app2` matches with the lhs of the current goal –i.e. $app(\underline{::_1\ (X,Y)}, Z)$ matches with $app(\underline{::_5\ (h,a)}, b)$ requires that $::_5 = ::_1$. These C-equations form an additional source justification the image of which must be satisfied in the target for a successful replay.

Deviations from a Simple Replay. As already described briefly, the main body of the analogical procedure replays the source plan node by node, checking justifications at each stage. There are three occasions when ABALONE deviates from this simple node by node replay and performs different types of adaptation:
First, peculiarities of the mappings trigger reformulations (Melis a) of the source plan. These reformulations are needed because sometimes the mappings alone are not sufficient to produce a correct plan proving the target

[6] We have omitted some irrelevant indices from Figure 1.13 for the sake of clarity. In general, however, all function symbols are indexed.

theorem. Reformulations do more than just map symbols. In general, reformulations may insert, replace or delete operators or may change operators, sequents and justifications of proof plan nodes in a substantiated rather than in an ad hoc way. In the example, a reformulation duplicates the WAVE (`app2`) before actually replaying it to WAVE(`plus2`) because a C-equation failed in a particular way.

Secondly, in the situation that a justification does not hold in the target, ABALONE will try to establish the justification. Its exact action will depend on the type of justification:

- If the failed justification is associated with WAVE, and the situation is such that a source rewrite-rule has no corresponding rule in the target, then ABALONE speculates a target rewrite-rule. It does this by applying m_b, m_e, and C-equations to the source rewrite-rule. In the example in Figure 1.13, the justification at the WAVE(lenapp2) node fails because there is no target image for the source rewrite-rules `lenapp2`. The appropriate action is to suggest a target rewrite-rules `lemma` by using the mappings and the C-equations to suggest a target rule. In the example, it uses the mappings $s_1 \mapsto s(w_1)$, $::_5 \mapsto s(s(w_2))$, and the C-equations $s_2 = s_1$ and $::_4 = ::_5$ to come up with the target rewrite-rule `lemma`: $half(X+s(s(Z))) \Rightarrow s(half(X+Z))$
- A justification may fail because some side-condition does not hold. Whereas the side-condition may trivially hold in the source, the mapped version in the target may not hold trivially. Hence, ABALONE will set up the target side-condition as a lemma.

If a justification does not hold and cannot be established, then ABALONE produces a target plan node that has an empty operator slot and a subgoal that contains *gap variables* $?_i$. The gap variable is a place holder for the unknown subexpression of the sequent in the (current) target node that corresponds to the source subexpression that was changed by the source operator which could not be transferred.

Thirdly, after the replay open goals are treated by the generative proof planner *CLAM*.

1.8.4 Summary and Results

ABALONE replays source *decisions*, among them decisions as difficult as the choice of induction schemes and induction variables for the INDUCTION operator. By replaying justifications, the system is flexible enough to suggest lemmata, override heuristics, and to override the default configuration of the generative planner.

ABALONE is implemented in Quintus Prolog as an extension to *CLAM*. It has been tested on a wide range of examples. It can plan several theorems that could not otherwise be planned in *CLAM* or in other provers fully automatically, see e.g., Melis and Whittle ().

Analogy-driven proof plan construction uses an extended derivational analogy and, in this sense, is similar to Prodigy/Analogy and PARIS. Abalone's use of abstraction differs from other systems. It uses abstractions at the meta-level only (Melis b). That is, the abstraction is used to restrict choises in the analogy process, and the retrieved cases are not abstracted, whereas in Paris abstract cases are stored, retrieved and refined. As opposed to most CBR systems, Abalone uses reformulations that are determined during the retrieval.

1.9 Summary and Related Work

On the one hand the four systems presented in this chapter have in common the synergistic integration of planning from first principles with a case-based reasoner. On the other hand they differ in many respects because of different motivations and mechanism.

1.9.1 Retain

Once a plan has been generated, each system performs different steps during the storage phase. In Prodigy/Analogy, an analysis of the solution is made relative to the problem description. Based on the partially ordered solution plan, connected components are determined and the corresponding interacting goals are identified to index independent subparts of the case. In addition, a goal regression process is performed to discriminate the initial state features relevant to the particular plan found. This process, known as the foot-printing process, ensures that only relevant features of the state cases are taken into account during retrieval. In Prodigy/Analogy, a case consists of a compression of the planning search episode. The case includes justifications of failed decisions taken during problem solving.

CAPlan/CbC computes the dependencies of the obtained solution. That is, the partial order in which the goals were achieved relative to the particular solution are computed and used as top indexing criteria of the new case. In addition, the complete goal graph structure is stored including valid and invalid decisions and their justifications.

Paris computes several abstractions of the found solution. These abstractions are stored as cases in the case base. The rationale behind storing several abstractions of the same solution is based on two facts: first, an abstracted solution at a higher level represents more concrete solutions than an abstracted solution at a lower level. Refining an abstract solution is more expensive the higher the level of the abstract solution. By storing several abstractions of the same solution Paris finds a balance between these two opposite facts.

In Abalone, the proof of a theorem is stored together with a description of the theorem which is abstracted to rippling paths representing a class of theorems having similar proofs.

1.9.2 Retrieval

PRODIGY/ANALOGY supports multi-case retrieval; cases covering disjoint subsets of the goals of the new problem are retrieved. The structure of the case-base in PRODIGY/ANALOGY reflects this principle by discriminating cases by the interacting top level goals. When matching the initial states of the candidate cases and the problem, only relevant features are taken into account. That is, only features that contribute to the particular solution of the case are considered as the relevance of a feature depends on the particular solution found. Retrieval is bounded to finding a case with a "reasonable" partial match.

In CAPLAN/CBC a dependency-driven retrieval strategy is performed. The idea is to consider interactions between the goals of the new problem, called dependencies. CAPLAN/CBC retrieves cases by considering the dependencies between the goals, first. This is reflected in the structure of the case base, as cases are discriminated by their dependencies at the top level. This strategy is adequate for domains, like process planning, in which interactions between the goals can be predetermined.

In PARIS a case corresponding to an abstract plan is retrieved. This plan solves the abstracted problem description of the current problem. The structure of the case base allows PARIS to find an appropriate abstract case at the lowest possible level of abstraction. In this way the adaptation costs are considered during retrieval as the lower the level is, the less effort is required for the refinement of the abstract case. Furthermore, PARIS makes use of a case deletion policy in order to avoid swamping of the case base.

In ABALONE, retrieval is a two-stage process. In the first stage the abstraction of source theorems is matched with the target theorem. An abstraction represents a whole class of theorems. In the second stage an element is retrieved from the class selected in stage one. This is done by searching for a second-order match between rules of the target problem with rules of source problems from the class. The search is supported by heuristic preference criteria. The second stage is comparable to searching for a case by matching the initial states.

1.9.3 Adaptation

PRODIGY/ANALOGY annotates the derivational trace of the cases with justifications. During replay, these justifications are verified in the new situation. In this way, the first-principles reasoner will not make decisions known from the case to be wrong. The justifications are expressed in a language describing situations occurring in state-space planning. A similar principle is followed in CAPLAN/CBC and ABALONE. However, given that their first-principles planners are different, the way the justifications are expressed and handled is different.

A case in CAPlan/CbC contains the goal graph expressing the relations between goals and decisions (valid and invalid). Justifications are expressed in these graphs and can be interpreted in terms of plan-space planning. When the goal graph is reconstructed relative to the new situation, the justifications are checked. The result, as in Prodigy/Analogy is that decisions known to be invalid from the case are not explored. However, by reconstructing the goal graph, CAPlan/CbC enables the user to interact with the first-principles planner CAPlan after replay has taken place.

Abalone's justifications formalize concepts in theorem proving such as identity of sorts, identity of function occurrences, and the existence of lemmata to be used. A singularity of the justifications in Abalone is that the validity of a justification can be possibly achieved by reformulations or by introducing new subgoals (lemmata). This is different from Prodigy/Analogy and CAPlan/CbC, where their justifications are matched directly in the current situation. nother unique ingredient of analogy-driven proof plan construction is reformulations that can be applied to enable a match of problems in the first place. These reformulations change a problem and its proof plan at the same time.

A case in Paris is an abstraction of a plan. As a result, there are no justifications because the search space of cases are expressed from the search space of the first-principles planners. In this context, adaptation refines the retrieved, abstract case to a solution at the concrete planning level.

1.9.4 Revise

Another common characteristic of the four case-based reasoners presented this chapter is that they are based on a first-principles reasoner that ensures that the plans obtained are always correct with respect to the domain theory. This contrasts to, say, Chef were the obtained solutions need to be revised as there is no guarantee of their validity.

1.9.5 Related Case-Based Planning Systems

Several other case-based planning systems have been developed in the USA and in Europe.

priar (Kambhampati 1994) reuses plans produced by a generative nonlinear hierarchical planner. Following the derivational analogy philosophy, priar uses the *validation structure* of a plan, which represents the dependencies among the plan steps, for retrieval and reuse of planning cases, but additionally employs domain independent strategies for plan adaptation. However, priar does not address the problem of how to organize a case base efficiently.

Within a deductive framework for planning, Köhler (1996) developed an approach to reuse plans. Her work concentrates on the retrieval process and used description logics as query languages for the plan library. In contraposition to the systems presented here, her MRL system requires the domain to be represented in formal logic. Further, for indexing the cases a formalization in terminological logic is also required.

The adaptation approaches described by Smyth and Keane (1993) and Lieber and Napoli () are similar to the use of reformulations in Abalone. The closest related system in theorem proving, Plagiator (Kolbe and Walther), differs from Abalone in many respects. Its pure transformational reuse produces a set of target assumptions rather than a target proof plan. For additional discussion of the differences see Melis and Whittle ().

MPA is a case-based planner performing multi-plan adaptation of partial-order plans (Francis and Ram 1995). Adaptation is made by transforming the derivational trace of the retrieved cases with a generic procedure instead of a generic planner as in the systems presented here. This procedure, however, is based on a careful study of the way the first-principles planner SNLP works. Thus, the procedure subsumes the work done by the generic planner.

1.10 Conclusion and Future Tasks

Several case-based planning systems (including the ones reported here) address important problems that occur in real planning applications. Experimental investigations have shown a drastic speedup (factor 1000 and more) of problem solving through case-based planning compared to pure generative planning approaches in many domains. However, several important questions have to be investigated in the future:

- knowledge engineering for case-based planning,
- user interaction, particularly during the reuse phase,
- maintaining the quality of plans (e.g., the cost and resource usage during plan execution) during reuse, and finally
- the integration of a case-based planner into an industrial environment.

We think that further, basic research in the area of case-based planning should try to overcome the following limiting assumptions because in several real world applications they do not hold.

- All required knowledge (e.g., operators, problem description, ...) is available prior to the planning process.
- Planning and execution are separated, i.e., first a plan is constructed and then the plan is executed.

Consider, for example, the task of planning an information gathering process from different sources. In this situation, several actions (e.g., operators for obtaining some information) must be executed prior to completing the plan. That is, it is necessary to produce a preliminary partial plan before all required information is available and to execute parts of the plan in order to get the information for completing the plan. First steps towards a model of planning that goes into this direction can be found in (Etzioni and Weld 1994; Knoblock).

Acknowledgments This work was supported by the NATO grant CRG.950405 and by the Deutsche Forschungsgemeinschaft SFB 378. This work was also supported by the Commission of European Communities (ESPRIT contract No. 22196, the INRECA II project, *Information and Knowledge Reengineering for Reasoning from Cases*). The partners of INRECA II are Acknosoft (prime contractor, France), Daimler-Benz (Germany), tecInno (Germany), Irish Multimedia Systems (Ireland), and University of Kaiserslautern (Germany).

References

Aamodt, A. and E. Plaza (1994). Case-based reasoning: foundational issues, methodological variations, and system approaches. *AI Communications 7*(1), 39–59.

Barrett, A. and D. Weld (1994). Partial-order planning: Evaluating possible efficiency gains. *Artificial Intelligence 67*(1), 71–112.

Bergmann, R. (1996). *Effizientes Problemlösen durch flexible Wiederverwendung von Fällen auf verschiedenen Abstraktionsebenen.* Ph. D. thesis, Universität Kaiserslautern. Available as DISKI 138, infix Verlag.

Bergmann, R., G. Pews, and W. Wilke (1993). Explanation-based similarity: A unifiying approach for integrating domain knowledge into case-based reasoning for diagnosis and planning tasks. See Wess, Althoff, and Richter (1993), pp. 182–196.

Bergmann, R. and W. Wilke (1995a). Building and refining abstract planning cases by change of representation language. *Journal of Artificial Intelligence Research 3*, 53–118.

Bergmann, R. and W. Wilke (1995b). Learning abstract planning cases. In N. Lavrač and S. Wrobel (Eds.), *Machine Learning: ECML-95, 8th European Conference on Machine Learning, Heraclion, Greece, April 1995*, Number 912 in Lecture Notes in Artificial Intelligence, pp. 55–76. Berlin: Springer.

Bergmann, R. and W. Wilke (1996). On the role of abstraction in case-based reasoning. Lecture Notes in Artificial Intelligence, 1186, pp. 28–43. Springer Verlag.

Bundy, A., F. van Harmelen, J. Hesketh, and A. Smaill (1991). Experiments with proof plans for induction. *Journal of Automated Reasoning 7*, 303–324.

Bylander, T. (1991). Complexity results for planning. In J. Mylopoulos and R. Reiter (Eds.), *Proceedings of the 12th International Conference on Artificial Intelligence IJCAI-91*, pp. 274–279.

Carbonell, J. G. (1983). Derivational Analogy and Its Role in Problem Solving.

Carbonell, J. G. (1986). Derivational analogy: a theory of reconstructive problem solving and expertise acquisition. In R. Michalski, J. Carbonnel, and T. Mitchell (Eds.), *Machine Learning: an Artificial Intelligence Approach*, Volume 2, pp. 371–392. Los Altos, CA: Morgan Kaufmann.

Carbonell, J. G., C. A. Knoblock, and S. Minton (1991). Prodigy: An integrated architecture for planning and learning. In K. VanLehn (Ed.), *Architectures for Intelligence*, pp. 241–278. Lawrence Erlbaum Associates, Publishers.

Etzioni, O. (1993). Acquiring search-control knowledge via static analysis. *Artificial Intelligence 62*, 255–301.

Etzioni, O. and D. Weld (1994). A softbot-based interface to the internet. *Comm. of ACM*.

Fikes, R. E. and N. J. Nilsson (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence 2*, 189–208.

Francis, A. G. and A. Ram (1993, July). The utility problem in case-based reasoning. In D. B. Leake (Ed.), *Case-Based Reasoning: Papers from the 1993 Workshop*, Technical Report WS-93-01, Menlo Park, CA, pp. 168. AAAI.

Francis, A. G. J. and A. Ram (1995). A domain-independent algorithm for multiplan adaptation and merging in least-commitment planning. In D. Aha and A. Rahm (Eds.), *AAAI Fall Symposium: Adaptation of Knowledge Reuse*, Menlo Park, CA. AAAI Press.

Hammond, K. J. (1986). CHEF: A Model of Case-Based Planning. pp. 267–271.

Holte, R. C., T. Mkadmi, R. M. Zimmer, and A. J. MacDonald (1995). Speeding up problem solving by abstraction: A graph-oriented approach. Technical report, University of Ottawa, Ontario, Canada.

Hutter, D. (1994). Synthesis of induction orderings for existence proofs. In A. Bundy (Ed.), *Proceedings of 12th International Conference on Automated Deduction (CADE-12)*, Lecture Notes in Artificial Intelligence 814, pp. 29–41. Springer.

Kambhampati, S. (1994). Expoiting causal structure to control retrieval and refitting during plan reuse. *10*(2), 213–244.

Kambhampati, S., L. Ihrig, and B. Srivastava. A candidate set based analysis of subgoal interactions in conjunctive goal planning. pp. 125–133.

Knoblock, C. Building a planner for information gathering: A report from the trenches. pp. 134–141.

Köhler, J. (1996). Planning from second principles. *Artificial Intelligence 87*.

Kolbe, T. and C. Walther. Second-order matching modulo evaluation – a technique for reusing proofs.

Lieber, J. and A. Napoli. Using classification in case-based reasoning. pp. 132–136.

McAllester, D. and D. Rosenblitt. Systematic nonlinear planning. pp. 634–639.

Melis, E. A model of analogy-driven proof-plan construction. pp. 182–189.

Melis, E. Solution-relevant abstractions constrain retrieval and adaptation.

Melis, E. (1994). How mathematicians prove theorems. In *Proc. of the Annual Conference of the Cognitive Science Society*, Atlanta, Georgia U.S.A., pp. 624–628.

Melis, E. and J. Whittle. Analogy in inductive theorem proving.

Melis, E. and J. Whittle (1997). Analogy as a control strategy in theorem proving. In *Proceedings of the 10th Florida International AI Conference (FLAIRS-97)*. also published as DAI Research Paper 840, University of Edinburgh, Dept. of AI.

Muñoz-Avila, H., J. Paulokat, and S. Wess. Controlling a nonlinear hierarchical planner using case replay. In *Advances in Case-Based Reasoning: Second European Workshop, EWCBR-94*, pp. 266–279.

Muñoz-Avila, H. and J. Hüllen. Retrieving relevant cases by using goal dependencies.

Muñoz-Avila, H. and F. Weberskirch. Planning for manufacturing workpieces by storing, indexing and replaying planning decisions.

Muñoz-Avila, H. and F. Weberskirch (1996). A specification of the domain of process planning: Properties, problems and solutions. Technical Report LSA-96-10E, Centre for Learning Systems and Applications, University of Kaiserslautern, Germany.

Petrie, C. (1991). *Planning and Replanning with Reason Maintenance*. Ph. D. thesis, University of Texas at Austin, Computer Science Dept.

Russell, S. and P. Norvig (1995). *Artificial Intelligence: A modern approach*. Englewood Cliffs, New Jersey: Prentice-Hall.

Sacerdoti, E. (1974). Planning in a hierarchy of abstraction spaces. *Artificial Intelligence 5*, 115–135.

Schank, R. C. (1982). *Dynamic Memory: A Theory of Learning in Computers and People*. New York: Cambridge University Press.

Smyth, B. and M. T. Keane. Remembering to forget. pp. 377–382.

Smyth, B. and M. T. Keane (1993). Retrieving adaptable cases: The role of adaptation knowledge in case retrieval. See Wess, Althoff, and Richter (1993), pp. 209–220.

Tambe, M. and A. Newell (1988). Some chunks are expensive. In *Proceedings of the 5th International Conference on Machine Learning*, pp. 451–458.

Veloso, M. M. (1994). *Planning and Learning by Analogical Reasoning*. Number 886 in Lecture Notes in Computer Science. Berlin: Springer.

Veloso, M. M. and J. Blythe (1994). Linkability: Examining causal link commitments in partial-order planning.

Waldinger, R. (1977). Achieving several goals simultaneously. *Machine Intelligence 8*, 94–136.

Weberskirch, F. (1995). Combining SNLP-like planning and dependency-maintenance. Technical Report LSA-95-10E, Centre for Learning Systems and Applications, University of Kaiserslautern, Germany.

Wess, S., K.-D. Althoff, and M. M. Richter (Eds.) (1993). *Topics in Case-Based Reasoning. Proc. of the First European Workshop on Case-Based Reasoning (EWCBR-93)*, Lecture Notes in Artificial Intelligence, 837. Springer Verlag.

Whittle, J. (1995). Analogy in *CL^AM*. Technical Report MSc.thesis, University of Edinburgh, Dept. of AI, Edinburgh.