CMU-CS-85-115

Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition

Jaime G. Carbonell
Computer Science Department,
Carnegie-Mellon University,
Pittsburgh, PA 15213

5 March 1985

Abstract

Derivational analogy, a method of solving problems based on the transfer of past experience to new problem situations, is discussed in the context of other general approaches to problem solving. The experience transfer process consists of recreating lines of reasoning, including decision sequences and accompanying justifications, that proved effective in solving particular problems requiring similar initial analysis. The role of derivational analogy in case-based reasoning and in automated expertise acquisition is discussed.[1]

---

[1] This research was supported in part by the Office of Naval Research (ONR) under grant numbers N00014-79-C-0661 and N00014-82-C-50767, and in part by a grant from IBM. The author thanks the following colleagues for their enlightening discussions that helped to clarify the ideas presented in this chapter: Jon Doyle, Jill Larkin, Steven Minton, and Allen Newell.

Table of Contents

List of Figures

1. Introduction: The Role of Analogy in Problem Solving

The term "problem solving" in artificial intelligence has been used to denote disparate forms of intelligent action to achieve well-defined goals. Perhaps the most common usage stems from Newell and Simon's work [31] in which problem solving consists of selecting a sequence of operators (from a pre-analyzed finite set) that transforms an initial problem state into a desired goal state. Intelligent behavior consists of a focused search for a suitable operator sequence by analyzing the states resulting from the application of different operators to earlier states.[2] Many researchers have adopted this viewpoint [16, 35, 32].

However, a totally different approach has been advocated by McDermott [24] and by Wilensky [42, 43] that views problem solving as plan instantiation. For each problem posed there are one or more plans that outline a solution, and problem solving consists of identifying and instantiating these plans. In order to select, instantiate, or refine plans, additionals plans that tell how to instantiate other plans or how to solve subproblems are brought to bear in a recursive manner. Traditional notions of search are totally absent from this formulation. Some systems, such as the counterplanning mechanism in POLITICS [7, 4], provide a hybrid approach, instantiating plans whenever possible, and searching to construct potential solutions in the absence of applicable plans.

A third approach is to solve a new problem by analogy to a previously solved similar problem. This process entails searching for related past problems and transforming their solutions into ones potentially applicable to the new problem [33]. I developed and advocated such a method [8, 9] primarily as a means of bringing to bear problem solving expertise acquired from past experience. The analogical transformation process itself may require search, as it is seldom immediately clear how a solution to a similar problem can be adapted to a new situation.

A useful means of classifying different problem solving methods is to compare them in terms of the amount and specificity of domain knowledge they require.

- If no structuring domain knowledge is available and there is no useful past experience to draw upon, weak methods such as heuristic search and means-ends analysis are the only tools that can be brought to bear. Even in these knowledge-poor situations, information about goal states, possible actions, their known preconditions and their expected outcomes is required.
- If specific domain knowledge in the form of plans or procedures exists, such plans may be instantiated directly, recursively solving any subproblems that arise in the process.

---

[2] In means-ends analysis, the current state is compared to the goal state and one or more operators that reduce the difference are selected, whereas in heuristic search, the present state is evaluated in isolation and compared to alternate states resulting from the application of different operators (to states generated earlier in the search), and the search for a solution continues from the highest-rated state.

- If general plans apply, but no specific ones do so, the general plans can be used to reduce the problem (by partitioning the problem or providing islands in the search space). For instance, in computing the pressure at a particular point in a fluid statics problem, one may use the general plan of applying the principle of equilibrium of forces at the point of interest (the vector sum of the forces = 0). But, the application of this plan only reduces the original problem to one of finding and combining the appropriate forces, without hinting how that may be accomplished in a specific problem [6, 22].
- If no specific plans apply, but the problem resembles one solved previously, apply analogical transformation to adapt the solution of that similar past problem to the new situation. For instance, in some studies it has proven easier for students to solve mechanics problems by analogy to simpler solved problems than by appealing to first principles or by applying general procedures presented in a physics text [11]. As an example of analogy involving composite skills rather than pure cognition, consider a person who knows how to drive a car and is asked to drive a truck. Such a person may have no general plan or procedure for driving trucks, but is likely to perform most of the steps correctly by transferring much of his or her automobile driving knowledge. Would that we had robots that were so self-adaptable to new, if recognizably related tasks!

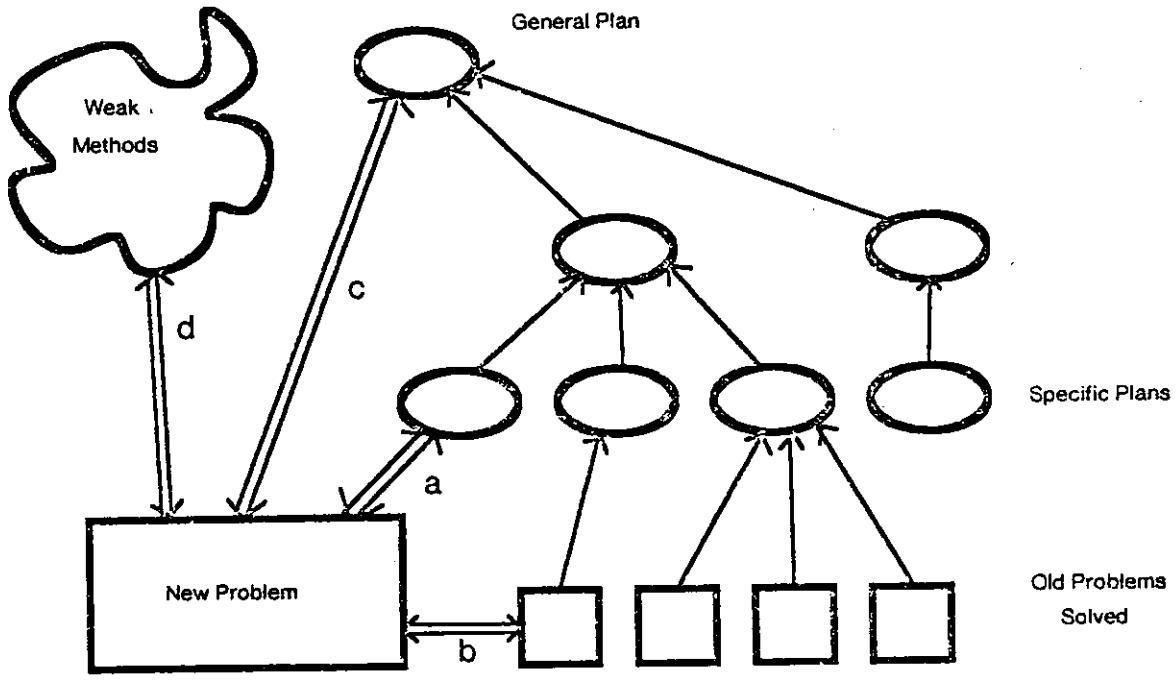


Figure 1-1: Problem solving may occur by a) instantiating specific plans, b) analogical transformation to a known solution of a similar problem, c) applying general plans to reduce the problem, d) applying weak methods to search heuristically for a possible solution, or e) a combination of these approaches.

Clearly, these problem solving approaches are not mutually exclusive; for instance, a "first-principles" approach can be used to reduce a problem to simpler subproblems, which can in turn be solved by analogy to recognizably similar past problems, or by any of the other methods. In fact, Larkin and I [6, 22] are developing a general inference engine for problem solving in the natural sciences that combines all four approaches.

As discussed earlier, only direct plan instantiation and weak methods have received substantial attention by AI practitioners. For instance, Newell and Laird's recent formulation of a universal weak method [21] as a general problem solving engine is developed completely within the search paradigm. Expert systems, for the most part, combine aspects of plan instantiation (often broken into small rule-size chunks of knowledge) and heuristic search in whatever manner best exploits the explicit and implicit constraints of the specific domain [15, 39, 14, 25, 26]. I am more concerned with the other two approaches, as they could conceivably provide powerful reasoning mechanisms not heretofore analyzed in the context of automating problem-solving processes, and allowing the problem solver to learn from experience. The rest of this chapter focuses on a new formulation of the analogical problem solving approach.

2. Analogy and Experiential Reasoning

The term *analogy* often conjures up recollections of artificially contrived problems asking: "X is to Y as Z is to ?" in various psychometric exams. This aspect of analogy is far too narrow and independent of context to be useful in general problem solving domains. Rather, I propose the following operational definition of analogical problem solving consistent with past AI research efforts [20, 44, 45, 17, 5, 9].

> **Definition:** *Analogical problem solving consists of transferring knowledge from past problem solving episodes to new problems that share significant aspects with corresponding past experience -- and using the transferred knowledge to construct solutions to the new problems.*

In order to make this definition operational, the problem solving method must specify:

- what it means for problems to "share significant aspects",
- what knowledge is transferred from past experience to the new situation,
- precisely how the knowledge transfer process occurs,
- and how analogically related experiences are selected from a potentially vast long term memory of past problem solving episodes.

There are two distinct approaches to analogical problem solving. The first approach, called *transformational analogy*, has been successfully implemented in ARIES (Analogical Reasoning and Inductive Experimentation System) [9]. The second approach, called *derivational analogy*, is a

reconstructive rather than transformational method, and is the topic of this paper. Both methods are analyzed with respect to the four criteria above.

2.1. Analogical Transformation of Past Solutions

If a particular solution has been found to work on a problem similar to the one at hand, perhaps it can be used, with minor modification, for the present problem. By "solution" I mean only a sequence of actions that if applied to the initial state of a problem brings about its goal state. Simple though this process may appear, an effective computer implementation requires that many difficult issues be resolved, to wit:

1. Past problems descriptions and their solutions must be remembered and indexed for later retrieval.
2. The new problem must be matched against large numbers of potentially relevant past problems to find closely related ones, if any. An operational similarity metric is required as a basis for selecting the most suitable past experiences.
3. The solution to a selected old problem must be transformed to satisfy the requirements of the new problem statement.

In order to achieve these objectives, the initial analogical problem solver [9] required a partial matcher with a built-in similarity criterion, a set of possible transformations to map the solution of one problem into the solution to a closely related problem, and a memory indexing mechanism based on a MOPS-like memory encoding of events and actions [37]. The solution transformation process was implemented as a set of atomic transform operators and a means-ends problem solver that searched for sequences of atomic transformations which when applied to the retrieved solution yielded potential solutions to the new problem. The resultant system, called ARIES, turned out to be far more complex than originally envisioned. Partial pattern matching of problem descriptions and searching in the space of solution transformations are difficult tasks in themselves. Figure 2-1 illustrates the transformational analogy process.

In terms of the four criteria, the solution transformation process may be classified as follows:

1. Two problems share significant aspects if they match within a certain preset threshold in the initial partial matching process, according to the built-in similarity metric.
2. The knowledge transferred to the new situation is the sequence of actions from the retrieved solution, whether or not that sequence is later modified in the analogical mapping process.
3. The knowledge transfer process is accomplished by copying the retrieved solution and perturbing it incrementally according to the primitive transformation steps in a heuristically guided manner until it satisfies the requirements of the new problem. (See [9] for details.)
4. The selection of relevant past problems is constrained by the memory indexing scheme and the partial pattern matcher.

Since a significant fraction of problems encountered in mundane situations and in areas requiring

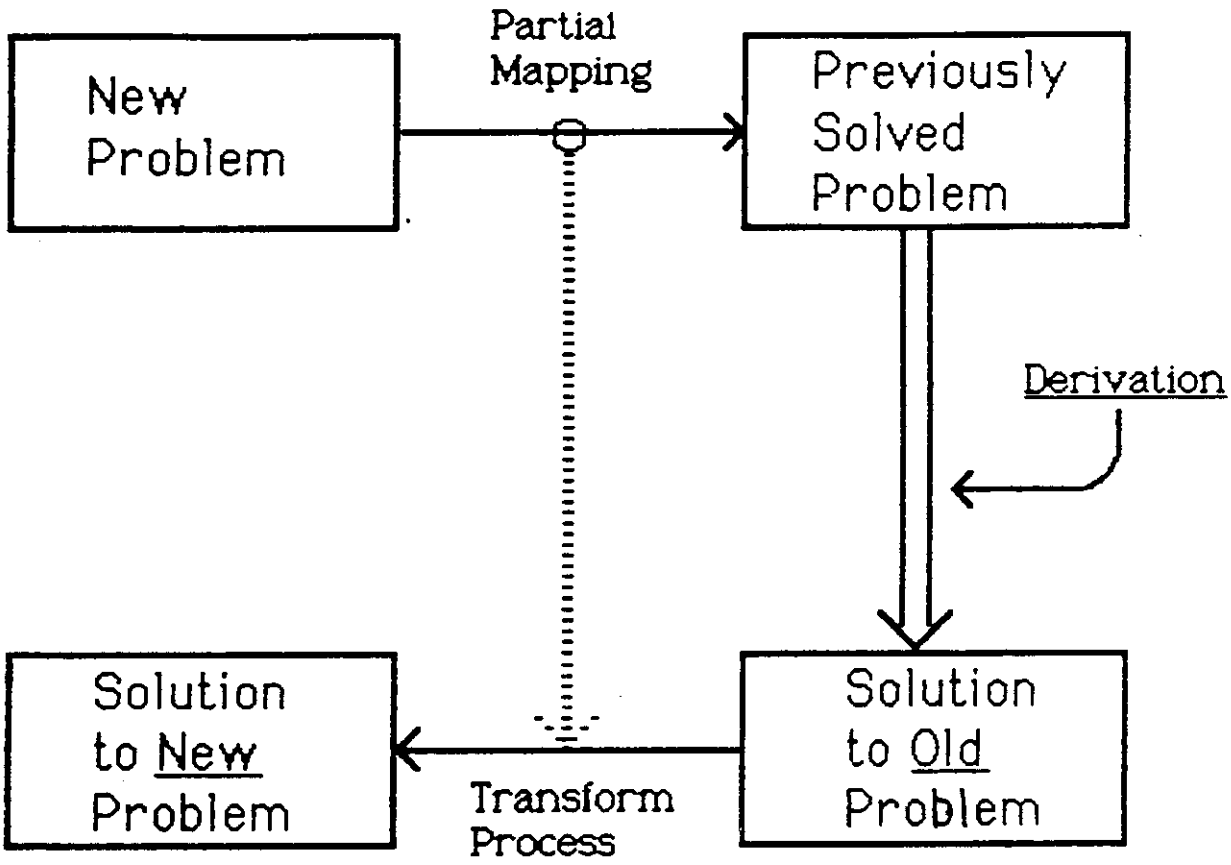


Figure 2-1: The transformational analogy process: Solutions to closely related problems are retrieved and modified to satisfy the requirements of the new problem.

substantial domain expertise (but not in abstract mathematical puzzles) bear close resemblance to past solved problems, the ARIES method proved effective when tested in various domains, including algebra problems and route planning tasks. An experiential learning component was added to ARIES that constructed simple plans (generalized sequences of actions) for recurring classes of problems, hence allowing the system to solve new problems in each recurrent class by the more direct plan instantiation approach. However, no sooner was the solution transformation method implemented and analyzed than some of its shortcomings became strikingly apparent. In response to these deficiencies, I started analyzing more sophisticated methods of drawing analogies, as discussed in the following sections.

3. The Derivational Analogy Method

In formulating plans and solving problems, a considerable amount of intermediate information is produced in addition to the resultant plan or specific solution. For instance, formulation of subgoal structures, generation and subsequent rejection of alternatives, and access to various knowledge structures are some of the intermediate steps in the problem solving process. But, the solution transformation method outlined above ignores all such information, focusing only upon the resultant sequence of actions and disregarding, among other things, the reasons for selecting those actions. Why should one take such extra information into account? It would certainly complicate the analogical problem solving process, but what benefits would accrue from such an endeavor? Perhaps the best way to answer this question is by analysis of where the simple solution transformation process falls short and how such problems may be alleviated or circumvented by preserving more information from which qualitatively different analogies may be drawn.

3.1. The Need for Preserving Derivation Histories

Consider, for instance, the domain of constructing computer programs to meet a set of pre-defined specifications. In the automatic programming literature, perhaps the most widely used technique is one of progressive refinement [3, 2, 19]. In brief, progressive refinement is a multi-stage process that starts from abstract specifications stated in a high level language (typically English or some variant of first order logic), and produces progressively more operational or algorithmic descriptions of the specification committing to control decisions, data structures and eventually specific statements in the target computer language. However, humans (well, at least this writer) seldom follow such a long painstaking process, unless perhaps the specifications call for a truly novel program unlike anything in one's past experience. Instead, a common practice is to recall similar past programs and reconstruct the new programming problem along the same directions. For instance, one should be able to program a quicksort algorithm in LISP quite easily if one has recently implemented quicksort in PASCAL. Similarly, writing LISP programs that perform tasks centered around depth-first tree traversal (such as testing equality of S-expressions or finding the node with maximal value) are rather trivial for LISP programmers but surprisingly difficult for those who lack the appropriate experience.

The solution transformation process proves singularly inappropriate as a means of exploiting past experience in such problems. A PASCAL implementation of quicksort may look very different than a good LISP implementation. In fact, attempting to transfer corresponding steps from the PASCAL program into LISP is clearly not a good way to produce any reasonable LISP program, let alone an elegant or efficient one. Although the two problem statements may have been similar, and the problem solving processes may preserve much of the inherent similarity, the resultant solutions (i.e., the PASCAL and LISP programs) may bear little if any direct similarities.

The useful similarities lie in the algorithms implemented and in the set of decisions and internal reasoning steps required to produce the two programs by successively refining the general specification of the algorithm. Therefore, the analogy must take place starting at earlier stages of the original PASCAL implementation, and it must be guided by a reconsideration of the key decisions in light of the new situation. In particular, the derivation of the LISP quicksort program starts from the same specifications, keeping the same divide and conquer strategy, but may diverge in selecting data structures (e.g. lists vs arrays), or in the method of choosing the comparison element, depending on the tools available in each language and their expected efficiency. However, future decisions (e.g. whether to recurse or iterate, what mnemonics to use as variable names, etc.) that do not depend on earlier divergent decisions can still be transferred to the new domain rather than recomputed. Thus, the derivational analogy method walks through the reasoning steps in the construction of the past solution and considers whether they are still appropriate in the new situation or whether they should be reconsidered in light of significant differences between the two situations.

The difference between the solution transformation approach and the derivational analogy approach just outlined can be stated in terms of the operational knowledge that can be brought to bear. The former corresponds to a person who has has never before programed quicksort and is given the PASCAL code to help him construct the LISP implementation, whereas the latter is akin to a person who has programmed the PASCAL version himself and therefore has a better understanding of the issues involved before undertaking the LISP implementation. Swartout and Balzer [40] and Scherlis [34] have argued independently in favor of working with program derivations as the basic entities in tasks relating to automatic programming. The advantages of the derivational analogy approach are quite evident in programming because the of the frequent inappropriateness of direct solution transformation, but even in domains whether the latter is useful, one can envision problems that demonstrate the need for preserving or reconstructing past reasoning processes.

3.2. The Process of Drawing Analogies by Derivational Transformation

Let us examine in greater detail the process of drawing analogies from past reasoning processes. Figure 3-1 depicts the process of mapping and merging past derivations to solve a new, analogically related problem. The essential insight is that useful experience is encoded in the reasoning process used to derive solutions to similar problems, rather than just in the resultant solution. And, a method of bringing that experience to bear in the problem solving process is required in order to make this form of analogy a computationally tractable approach. Here we outline such a method:

1. When solving a problem by whatever means, store each step taken in the solution process, as illustrated in figure 3-2, including:
    - The subgoal structure of the problem

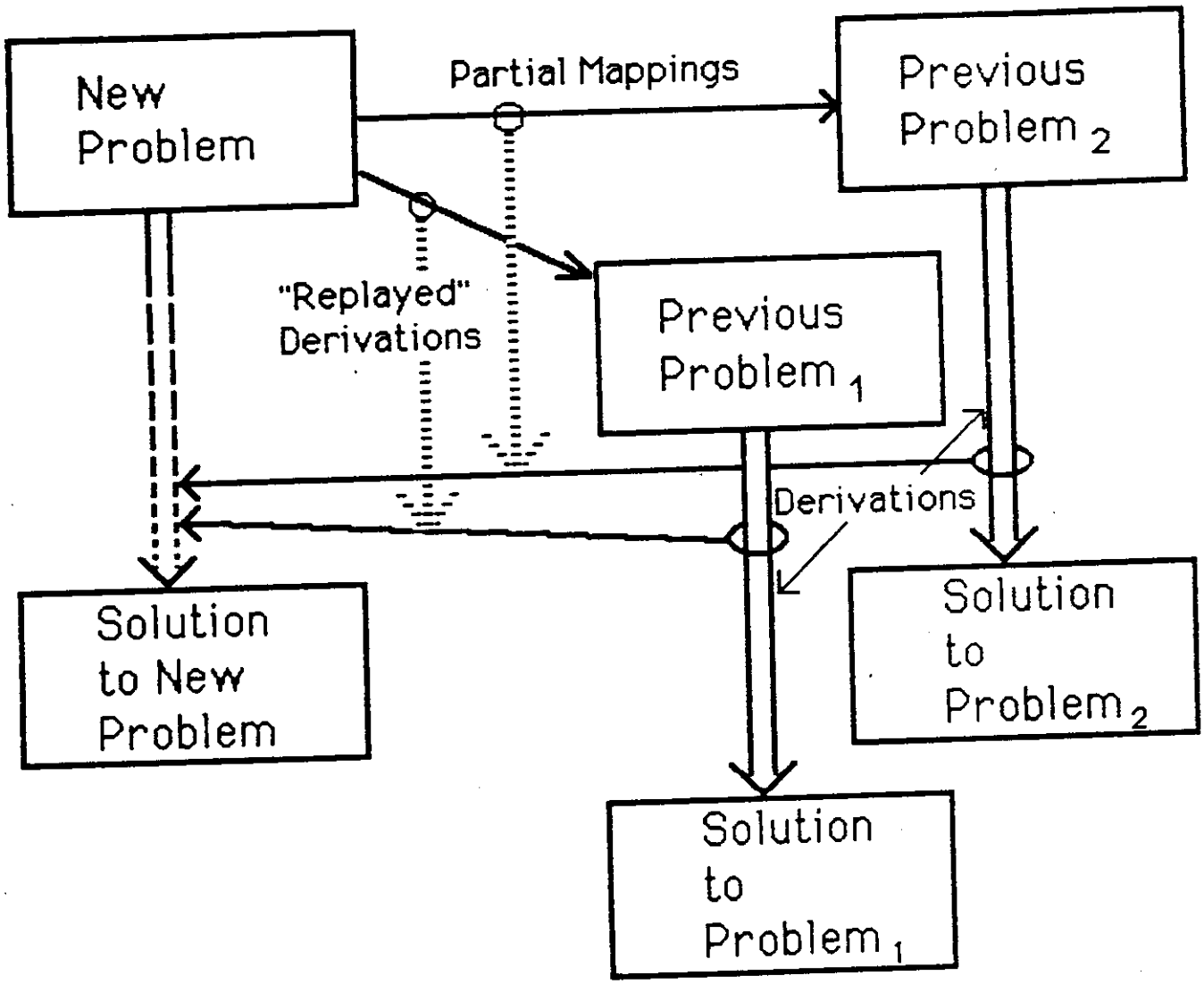


Figure 3-1: The transformational analogy process: The derivational traces of similar past problems are replayed, and where necessary modified, to reconstruct a solution to a similar new problem.

- Each decision made (whether a decision to take action, to explore new possibilities, or to abandon present plans), including:
  - Alternatives considered and rejected
  - The reasons for the decisions taken (with dependency links to the problem description or information derived therefrom)

- The start of a false path taken (with the reason why this appeared to be a promising alternative, and the reason why it proved otherwise, again with dependency links to the problem description. Note that the body of the false path and other resultant information need not be preserved.)
- Dependencies of later decisions on earlier ones in the derivation.

- Pointers to the knowledge that was accessed and proved useful in the eventual construction of the solution
- The resultant solution itself
  - In the event that the problem solver proved incapable of solving the problem, the closest approach to a solution should be stored, along with the reasons why no further progress could be made (e.g., a conjunctive subgoal that could not be satisfied).
  - In the event that the solution depends, perhaps indirectly, on volatile assumptions not stated in the problem description (such as the cooperation of another agent, or time-dependent states) store the appropriate dependencies.

2. When a new problem is encountered that does not lend itself to direct plan instantiation or direct recognition of a solution pattern, start to analyze the problem by applying general plans or weak methods, whichever is appropriate to the situation.
3. If after commencing the analysis of the problem, the reasoning process (the initial decisions made and the information taken into account) parallels that of past problem situations, retrieve the full reasoning traces and proceed with the derivational transformation process. If not, consider the possibility of solution transformation analogy or, failing that, proceed with the present line of non-analogical reasoning.

- Two problems are considered similar if their analysis results in equivalent reasoning processes, at least in its initial stages. This replaces the more arbitrary context-free similarity metric required for partial matching among problem descriptions in drawing analogies by direct solution transformation. Hence, past reasoning traces (henceforth *derivations*) are retrieved if their initial segment matches that the first stages of the analysis of the present problem. Or, to state it differently, problem solving episodes are judged to be derivationally similar if the state of the reasoner in the initial stages of solving the latter problem partially recreates the earlier internal state.
- The retrieved reasoning processes are then used much as individual relevant cases in medicine are used to generate expectations and drive the diagnostic analysis. Reasoning from individual cases has been recognized as an important component of expertise [38], but little has been said of the necessary information that each case must contain, let alone providing a simple method of retrieving the appropriate cases in a manner that does not rely on arbitrary similarity metrics. Here, I take the stand that cases must contain the reasoning process used to yield an answer, together with dependencies to the particular circumstances of the problem, pointers to data that proved useful, list of alternative reasoning paths not taken, and failed attempts (coupled with both reasons for their failure and reasons for having originally made the attempt). Case-based reasoning is nothing more than derivational analogy applied to domains of extensive expertise.
- It is important to know that although one may view derivational analogy as an interim step in reasoning from particular past experience as more general plans are acquired, it is a mechanism that remains forever useful, since knowledge is always incomplete and

exceptions to the best formulated general plans require representation and use of individual reasoning episodes.

4. A retrieved derivation is applied to the new problem as follows: For each step in the derivation, starting immediately after the matched initial segment, check whether the reasons for performing that step are still valid by tracing dependencies in the retrieved derivation to relevant parts of the old problem description or to volatile external assumptions made in the initial problem solving.

    - If parts of the problem statement or external assumptions on which the retrieved situation rests are also true in the present problem situation, proceed to check the next step in the retrieved derivation.
    - If there is a violated assumption or problem statement, check whether the decision made would still be justified by a different derivation path from the new assumptions or statements. If so, store the new dependencies and proceed to the next step in the retrieved derivation. The idea of tracing causal dependences and verifying past inference paths borrows heavily from TMS [12] and some of the non-monotonic logic literature [23]. However, the role played by data dependencies in derivational analogy is somewhat different and more constrained than in maintaining global consistency in deductive data bases.
    - If the old decision cannot be justified by new problem situation,
        - evaluate the alternatives not chosen at that juncture and select an appropriate one in the usual problem solving manner, storing it along with its justifications, or
        - initiate the subgoal of establishing the right supports in order for the old decision to apply in the new problem[3] (clearly, any problem solving method can be brought to bear in achieving the new subgoal), or
        - abandon this derivational analogy in favor of another more appropriate problem solving experience from which to draw the analogy or in favor of other means of problem solving.
    - If one or more failure paths are associated with the current decision, check the cause of failure and the reasons these alternatives appeared viable in the context of the original problem (by tracing dependency links when required). In the case that their reasons for failure no longer apply, but the initial reasons for selecting these alternatives are still present, consider reconstructing this alternate solution path in favor of continuing to apply and modify the present derivation (especially if quality of solution is more important than minimizing problem solving effort).
    - In the event that a different decision is taken at some point in the rederivation, do not abandon the old derivation, since future decisions may be independent of some past decisions, or may still be valid (via different justifications) in spite of the somewhat different circumstances. This requires that dependency links be kept between decisions at different stages in the derivation.
    - The derivational analogy should be abandoned in the event that a preponderance of the

---

[3] This approach only works if the missing or violated premise relates to that part of the global state under control of the problem solver, such as acquiring a missing tool or resource, rather than under the control of an uncooperative external agent or a recalcitrant environment. The discussion of strategy-based counterplanning gives a more complete account of subgoaling to rectify unfulfilled expectations [4, 7].

old decisions are invalidated in the new problem situation. Exactly what the perseverance threshold should be is a topic for empirical investigation, as it depends on whether there are other tractable means of solving this problem and on the overhead cost of reevaluating individual past decisions no longer be supported which may or may not have independent justification.

5. After an entire derivation has been found to apply to the new problem, store its divergence from the parent derivation as another potentially useful source of analogies, and as an instance from which more general plans can be formulated if a large number of problems of share a common solution procedure [9].

3.3. Efficiency Concerns

An important aspect of the derivational analogy approach is the ability to store and trace dependency links. It should be noted that some of the inherent inefficiencies in maintaining global consistency in a large deductive data base do not apply, as the dependency links are internal to each derivation with external pointers only to the problem description and to any volatile assumptions necessitated in constructing the resultant solution. Hence, the size of each dependency network is quite small, compared to a dependency network spanning all of memory. Dependencies are also stored among decisions taken at different stages in the temporal sequence of the derivation, thus providing the derivational analogy process access to causal relations computed at the time the initial problem was solved.

The analogical re-derivation process is not inherently space inefficient, although it may so appear at first glance. The sequence of decisions in the solution path of a problem are stored, together with necessary dependencies, the problem description, the resultant solution, and alternative reasoning paths not chosen. Failed paths are not stored, only the initial decision that was taken to embark upon that path, and the eventual reason for failure (with its causal dependencies), are remembered. Hence, the size of the memory for derivational traces is proportional to the depth of the search tree, rather than to the number of nodes visited. Problems that share large portions of their derivational structure can be so represented in memory, saving space and facilitating the similarity-based indexing process. Moreover, when a generalized plan is formulated for recurring problems that share a common derivational structure, the individual derivations that are totally subsumed by the more general structure can be permanently masked or deleted. Those derivations that represent exceptions to the general rule, however, are precisely the instances that should be saved and indexed accordingly for future problem solving [18].

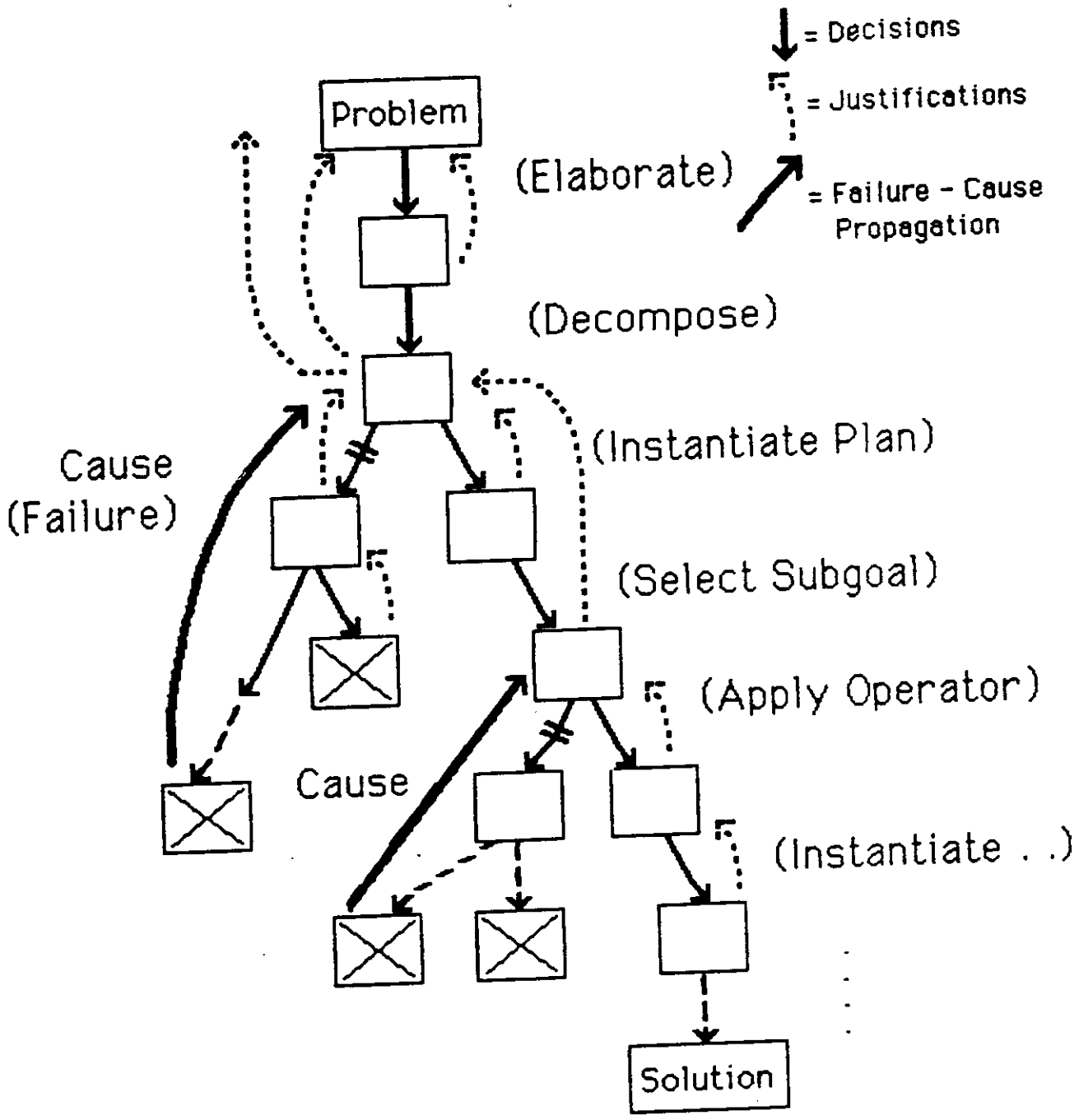


Figure 3-2: A derivational trace: Each reasoning step is justified in terms of previous reasoning steps or external knowledge. When a solution attempt fails, the cause of failure is propagated back to the branching point from the successful path and retained.

3.4. Summarizing the Derivational Process

Derivational analogy bears closer resemblance to Schank's reconstructive memory [36, 37] and Minsky's K-lines [28] than to traditional notions of analogy. Although derivational analogy is less ambitious in scope than either of these theories, it is a more precisely defined inference process that can lead to an operational method of reasoning from particular experiential instances. The key notion is to reconstruct the relevant aspects of past problem solving situations and thereby transfer knowledge to the new scenario, where that knowledge consists of decision sequences and their justifications, rather than individual declarative assertions. To summarize, consider how the process of derivational analogy can be described in terms of the for criteria for analogical reasoning presented in the previous section:

1. Two problems share significant aspects if their initial analysis yields the same reasoning steps, i.e., if the initial segments of their respective derivations start by considering the same issues and making the same decisions.
2. The justified steps in the derivation may transferred to the new situation, in essence recreating the significant aspects of the reasoning process that solved the past problem.
3. Knowledge transfer is accomplished by reconsidering old decisions in light of the new problem situation, preserving those that apply, and replacing or modifying those whose supports are no longer valid in the new situation.
4. Problems and their derivations are stored in a large episodic memory along the line of Schank's MOPS [37], and retrieval occurs by replication of initial segments of decision sequences recalling the past reasoning process.

4. Incremental Expertise Acquisition

Derivational analogy is a fertile computational paradigm that supports various knowledge acquisition and skill refinement strategies. Thus far, I have focused on the basic problem solving aspects, but a major motivation behind the reconstructive derivational strategy is the natural manner in which it can be extended to include incremental acquisition of domain expertise. First, let us dwell briefly upon case-based reasoning as a major component of human expertise. Then, let us turn to some concrete methods for acquiring and refining expertise from experience, based upon the derivational analogy model.

4.1. Case-Based Reasoning as a Model of Human Expertise

The vast majority of present-day expert systems encode their knowledge as a large, amorphous set of domain-specific rules [15, 26, 25, 39, 41]. The "knowledge engineering" task is defined as one of extracting from the human expert the set of rules that comprise his or her expertise in a particular, well-defined domain. The task is by no means easy; quite the contrary. It can take years of laborious efforts by teams of domain experts and AI researchers in an iterative process of formulating, evaluating, re-formulating, discarding and refining a set of rules to develop the knowledge base of a

particular expert system. Observing this phenomenon, Edward Feigenbaum uttered his now famous proclamation "In the knowledge lies the power". How right he was! Fortunately, however, the tacit assumption that domain knowledge must necessarily be represented as large sets of context-independent rules is proving to be only an early engineering decision, and a very limiting one at that. The knowledge must be captured, but the question remains as to the best means of acquiring and representing it in a computationally effective manner.

What then would be an alternative means of representing and acquiring domain knowledge? In order to address this question, I set out to build an expert system and gain first-hand experience, but keeping in perspective all the different problem solving methods and machine learning paradigms. In less than a year, with the help one programmer and two domain experts, we produced SMOKEY [10], a prototype fire diagnosis expert system. In essence, SMOKEY polls multiple remote sensors (heat, smoke, air-pressure detectors, etc.), and calculates the location, expected spread and critical nature of a fire on a building or a ship. From this assessment it recommends actions such as signaling safe exit routes free of smoke, closing down air-circulation ducts before they spread toxic smoke to unaffected areas, selecting equipment for the fire-fighting team appropriate to the nature of the fire, and so on. We learned several lessons from this endeavor, and here we focus on the central one: the utility of case-based reasoning.

Upon interviewing Naval experts for on-board fire diagnosis situations we found that for sizable fires, they are swamped with too much information coming from all the potentially relevant sensors. Hence, we played a video-tape simulation of several fires at much-reduced speeds. The results were amazing: previously sloppy decisions that ignored crucial information vanished, and we recorded elaborate problem-solving protocols including fairly complete justifications for each action or decision taken. Unfortunately, one cannot put real fires in slow motion to allow for human reaction time and memory limitations. Thus, the need for a SMOKEY-like system on a fast processor was established. Now, the question remained as to how the excellent (slow-motion) problem solving traces could be converted into the knowledge base of an expert system. And, the key insight was that perhaps they need not be converted -- only encoded appropriately and fed to a derivational analogy problem solver and learning module. SMOKEY was built concurrently with the development of the derivational analogy method, so the expertise acquisition steps discussed below were carried out largely by hand, rather than in a completely automated fashion.

We found that human experts are incredibly poor at producing general deductive rules that account for their behavior. When forced to do so by insistent knowledge engineers, they try hard and produce faulty rules. When later faced with a problem in which the rule fails, the typical response is: "Well, I

didn't think of *that* situation, but perhaps I can fix the rule...or add a new one..." This *ad-hoc* iterative process, slow and frustratingly inefficient as it may be, usually converges upon an acceptable knowledge base. However, a much more efficient and humane approach is to let the experts do what they do best: solve problems in their domain of expertise. The only added burden is a reporting requirement. Each problem solving step, including references to static domain knowledge or to heuristics of the domain, must be reported explicitly, along with the reason why such knowledge was used. This process provides external derivational traces that a derivational analogy inference engine can use to solve similar future problems in an effective manner. Although the derivational method was originally conceived as a means to reason and learn from one's own past experience, it works equally well as a means to reason and learn from the experience of a more knowledgeable external source, such as a human expert, or a worked out problem example in a text book.

Case-based reasoning is particularly prevalent in law -- at least in the British and American systems of jurisprudence -- and in medical diagnosis and treatment. The idea of case-based reasoning in expert systems is not new. Schank [38], for instance, advocates this method as superior and closer to human reasoning than present expert systems. Doyle [13] proposes the notion of emulating the human master-apprentice process as a means whereby the latter (human or computer) can acquire expertise by replicating the reasoning processes of former. Here, I propose a concrete computational mechanism -- the derivational analogy process -- as a means of providing expert systems with the ability to reason from cases, whether the cases be past experience or externally acquired knowledge. However, I also believe that human experts can solve problems progressively more quickly and effectively with repeated experience. Whereas case-based reasoning may reflect accurately a crucial intermediate stage in the learning process and may account for problem solving behavior in infrequently recurring situations, some knowledge is gradually compiled into more general processes abstracted from the concrete cases. That is to say, for the most routine, recurring problems, the derivational analogy process should produce general plans that can be instantiated directly. The following section explores learning techniques in derivational analogy.

4.2. Automatic Acquisition of Plans and Strategies

The standard behavioral definition for learning can be paraphrased as:

> **definition:** *A system (biological or mechanical) is said to learn if it can modify its behavior after a set of experiences such that it can perform a task more accurately or more efficiently than before, or it can perform a new task beyond its previous capabilities.*

What can be learned in the derivational analogy process, according to this definition? Learning can occur at many levels, and in many forms.

4.2.1. Enrichment of Case-Based Memory

As a system solves problems, or is presented with fully annotated derivations of solutions, its repertoire of cases increases. Thus, it will be able to derive analogical solutions from these new experiences. This incremental monotonic increase in its experiential knowledge base provides a powerful argument in favor of a method such as derivational analogy, which can utilize the experience directly to solve new problems.

4.2.2. Generalized Plans

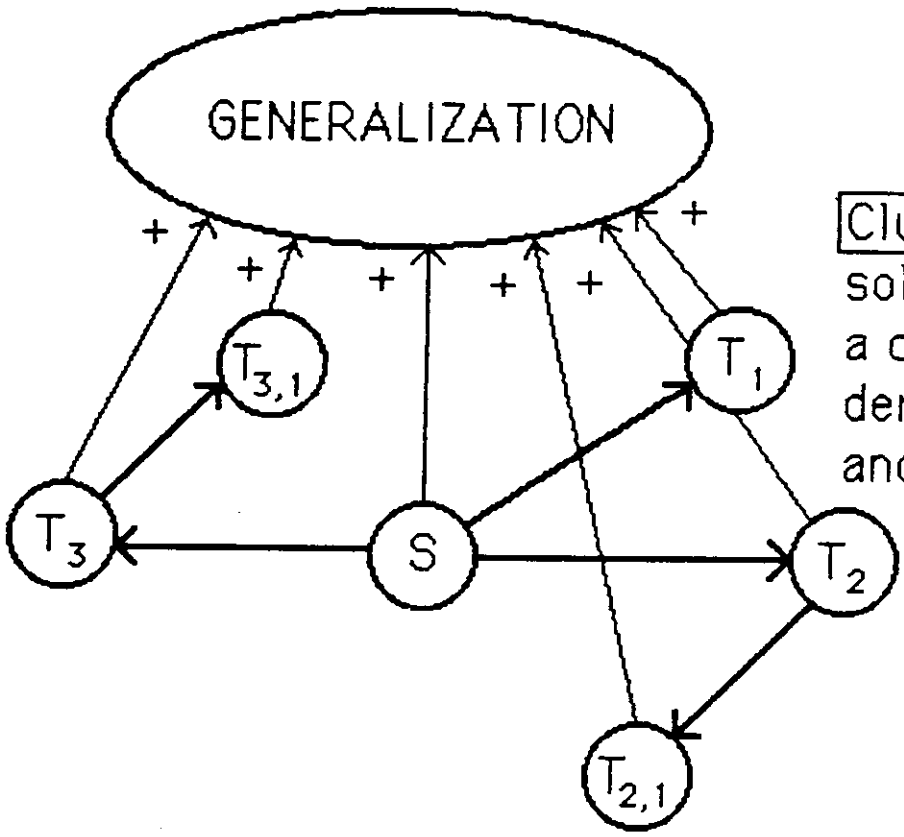
Using only the resultant solutions to a large set of analogically related problems (rather than the entire derivations), generalized plans can be abstracted. This process requires that solutions derived from a common analogical parent from a set of positive exemplars, and unrelated or failed solutions form a set of negative exemplars. These sets are given to a general inductive engine [27], or preferably an incremental one such as Mitchell's version space method [29, 30], which abstracts a generalized plan from the recurring common aspects of these solutions. Later, the generalized plan can be instantiated directly -- or refined further if more instance solutions are derived. Figure 4-1 summarizes this process, which is discussed at greater length in [9].

4.2.3. Strategy Acquisition

The same method for inducing generalized plans from positive and negative exemplars can be applied to different parts of the full derivational trace.

Considerations of alternate decision points in derivationally related solutions can yield to the compilation of domain specific heuristics for making future choices of the same nature. If a particular decision was part of a successful derivation in several problem solutions, but led to a false path under other problem solutions, we again have the requisite grist for the induction engine: A set of positive exemplars in the justifications of the successful decision, and a set of near-miss negative exemplars in the cases where the same decision proved ineffective. In fact, the cause of failure (propagated back to the causally related decision point and retained in the derivational trace, as discussed earlier) provides a set of necessary -- but perhaps not sufficient -- conditions to discriminate between the positive and negative instances of the decision.

Consider, for instance, the selection of a means of transportation in various problem solving situations that involve travel. If an automobile was successfully selected three times to travel between cities in the continental United States, but was erroneously suggested as a means of traveling between Boston and London, the strategy for selecting a means of transportation can be refined. The cause of failure (no land route between the source and destination) serves to add a necessary



$$\delta(T_i, T_j) << \delta(T_i, T_k)$$

$$\forall T_i, T_j T_k \text{ s.t. } T_i \in \mathbf{C} \longleftrightarrow T_j \in \mathbf{C} \;\&\; T_i \in \mathbf{C} \longleftrightarrow T_k \notin \mathbf{C}$$

- Members of a cluster = + Instances
- Members of other clusters = - Instances

(or failed analogies serve as - Instances)

to an Induction Engine

Figure 4-1: Generalizing plans from analogically related solutions: Solutions derived from a common transformational ancestors from a cluster of positive exemplars. Failed attempts, and members of other clusters provide the negative exemplars to an induction engine.

condition to the strategy, and the fact that automobile travel proved successful independent of the exact compass orientation or identity of the cities within the United States serves to assert the independence of the strategy from such considerations. In fact, a trial implementation in the route-finding domain has yielded planning strategies increasingly more appropriate to the task domain.

Applying the same technique to problem decomposition tasks, plan-selection tasks (when multiple generalized plans exist for a given subproblem), and avoiding the causes of failure under similar circumstances can also yield automated refinement of the system's behavior. In all cases, the internal and external justifications provide the means to focus on the functionally relevant aspects of the phase in the derivation from which the system is attempting to learn. However, unlike the strategy-selection task above, I have no empirical validation of the utility or feasibility of attempting to produce better problem decomposition criteria, plan selection methods, or generalized avoidance of recurring pitfalls. This is currently an active area of exploration.

4.2.4. Fractioning Derivations into Rules

A process akin to "decompilation" is the formulation of generally applicable rules from more problem specific derivational sequences. Contrary to Anderson [1] and others who view knowledge compilation as the perhaps the most significant learning strategy, I view the decompilation process to be at least as important. Recall that in case-based reasoning one is given compiled, but fully annotated, audit trails of the reasoning process -- the derivational traces. The fractioning task is one of axiomatizing the long, problem-specific traces into individual rules applicable to a much wider range of situations, although each rule solves only part of the new problem. The difficult aspect of the task is to bundle the derivationally-related steps into useful rules, assuring that the necessary (and only the necessary) preconditions are associated with each rule. But its utility lies in the ability to learn more generally applicable knowledge from specific experiences. The knowledge engineers may yet have their precious rule sets, but rule sets generated automatically after extended experience with derivational traces, rather than rules produced and gradually refined by hand at much cost in time and frustration.

Let us see how rules would be fractioned off from longer derivational traces. The process described below has been tested only in the route-finding domain thus far, but there it has proven useful.

1. ***Find relevant candidates*** -- The first step in the formulation of rules from derivational traces is to search for candidate subsequences of actions that recur in different, possibly unrelated, derivational traces in the domain. For instance, the sequence:

```
LOCATE(bridge),
PLAN-ROUTE(here,bridge),
PLAN-ROUTE(bridge,destination)
```

Occurred with high frequency, and was proposed as a rule kernel.

2. ***Trace justifications*** -- Why must one plan a route, or locate a bridge? The justification for the former comes from the supergoal goal `PLAN-ROUTE(here,destination)`, and the justification for the latter comes from the fact that the presence of a river between "here" and "destination" violates a precondition for land travel.

3. ***Formulate rule*** -- First, computable predicates must be found to establish the justifications. These become the condition side of the rule. Then, the justified subsequence of actions, parameterized to the most general justifiable class of actions or objects becomes the action side of the rule. In the present example, the resultant rule is:

```
If GOAL(x) is LOC(x,time-2) = destination
 & LOC(x,time-1) = here
 & BETWEEN(here,destination) = river
 & TRANSPORTATION(x) = land-vehicle
Then FIND(bridge,river)
     PLAN-ROUTE(here,bridge)
     PLAN-ROUTE(bridge,destination)
```

Thus, we see that from multiple planning episodes, one can induce the rule that if one must cross a river, then one should first worry about finding a bridge, and then plan the route according to this constraint. The rule fractioning process truly requires all three phases: finding relevant sequences, determining the justifications for these sequences, and actually formulating the rule from this information. Without a derivational trace, it would not be possible to fraction rules reliably, because the justifications provided in the trace are needed for searching out the *necessary and useful* conditions for the left-hand-side of the rule. Otherwise, one would have to either postulate that the recurrent subsequence was totally independent of context (a terrible assumption -- the system would be searching for bridges when there were no rivers to cross), or completely dependent on context, requiring that the entire trace up to that point be included in the condition side, rather than just the causally relevant conditions indicated by the justifications.

5. Concluding Remark

Derivational analogy is a powerful reasoning mechanism, and one that provides the necessary information for learning to occur in many different forms, from accumulation of cases to formulation of domain-oriented strategies and sets of deductive rules. It has been remarked that heuristics are "compiled hindsight", and as such can prove useful in guiding future behavior. But, how can one take advantage of hindsight unless one recalls past experiences including aspects of one's state of mind necessary to reconstruct past problem-solving behaviors in new situations? There must be a retrospective process able to exploit past experience, and a gradual, incremental learning process that abstracts from that experience more generally applicable chunks of knowledge. The derivational analogy process is one concrete method for realizing the former, and the strategy and rule acquisition

processes are means of implementing the latter. Together they form a computational theory of incremental expertise acquisition, a theory that is still in the process of being implemented, tested, refined, and reformulated.

6. References

1. Anderson, J. A., "Acquisition of Proof Skills in Geometry," in *Machine Learning, An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell and T. M. Mitchell, eds., Tioga Press, Palo Alto, CA, 1983.

2. Balzer, R., "Imprecise Program Specification," Tech. report RR-75-36, USC/Information Sciences Institute, 1975.

3. Barstow, D. R., *Automatic Construction of Algorithms and Data Structures Using a Knowledge Base of Programming Rules*, PhD dissertation, Stanford University, Nov. 1977.

4. Carbonell, J. G., "Counterplanning: A Strategy-Based Model of Adversary Planning in Real-World Situations," *Artificial Intelligence*, Vol. 16, 1981, pp. 295-329.

5. Carbonell, J. G., "A Computational Model of Problem Solving by Analogy," *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, August 1981, pp. 147-152.

6. Carbonell, J. G., Larkin, J. H. and Reif, F., "Towards a General Scientific Reasoning Engine," Tech. report, Carnegie-Mellon University, Computer Science Department, 1983, CIP # 445.

7. Carbonell, J. G., *Subjective Understanding: Computer Models of Belief Systems*, Ann Arbor, MI: UMI research press, 1981.

8. Carbonell, J. G., "Experiential Learning in Analogical Problem Solving," *Proceedings of the Second Meeting of the American Association for Artificial Intelligence*, Pittsburgh, PA, 1982.

9. Carbonell, J. G., "Learning by Analogy: Formulating and Generalizing Plans from Past Experience," in *Machine Learning, An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell and T. M. Mitchell, eds., Tioga Press, Palo Alto, CA, 1983.

10. Carbonell, J. G., "The SMOKEY Fire-Diagnosis System," Tech. report, Carnegie-Mellon University, Computer Science Department, 1985.

11. Clements, J., "Analogical Reasoning Patterns in Expert Problem Solving," *Proceedings of the Fourth Annual Conference of the Cognitive Science Society*, 1982.

12. Doyle, J., "A Truth Maintenance System," *Artificial Intelligence*, Vol. 12, 1979, pp. 231-272.

13. Doyle, J., "Expert Systems Without Computers," *AI Magazine*, Vol. 5, No. 2, 1984, pp. 59-63.

14. Duda. R. O., Hart, P. E., Konolige, K. and Reboh, R., "A Computer-Based Consultant for Mineral Exploration," Tech. report 6415, SRI, 1979.

15. Feigenbaum, E. A., Buchanan, E. A. and Lederberg, J., "On Generality and Problem Solving: A Case Study Using the DENDRAL Program," in *Machine Intelligence 6*, D. Michie, ed., Edinburgh University Press, 1971.

16. Fikes, R. E. and Nilsson, N. J., "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence*, Vol. 2, 1971, pp. 189-208.

17. Gentner, D., "The Structure of Analogical Models in Science," Tech. report 4451, Bolt Beranek and Newman, 1980.

18. Hayes-Roth, F., "Using Proofs and Refutations to Learn from Experience," in *Machine Learning, An Artificial Intelligence Approach,* R. S. Michalski, J. G. Carbonell and T. M. Mitchell, eds., Tioga Press, Palo Alto, CA, 1983.

19. Kant, E., *Efficiency in Program Synthesis,* UMI Research press, Ann Arbor, MI, 1981.

20. Kling, R. E., "A Paradigm for Reasoning by Analogy," *Artificial Intelligence,* Vol. 2, 1971, pp. 147-178.

21. Laird, J. E. and Newell, A., "A Universal Weak Method," *Proceedings of the Eight Joint Conference on Artificial Intelligence,* 1983, (submitted).

22. Larkin, J., Reif, F. and Carbonell, J. G., "FERMI: A Flexible Expert Reasoner with Multi-Domain Inference," *Cognitive Science,* Vol. 9, 1984 (Submitted).

23. McDermott, D. V. and Doyle J., "Non-Monotonic Logic I," *Artificial Intelligence,* Vol. 13, 1980, pp. 41-72.

24. McDermott, D. V., "Planning and Acting," *Cognitive Science,* Vol. 2, No. 2, 1967, pp. 71-109.

25. McDermott, J., "R1: A Rule-Based Configurer of Computer Systems," Tech. report, Carnegie-Mellon University, Computer Science Department, 1980.

26. McDermott, J., "XSEL: A Computer Salesperson's Assistant," in *Machine Intelligence 10,* Hayes,-J., Michie, D. and Pao, Y-H., eds., Chichester UK: Ellis Horwood Ltd., 1982", pp. 325-337.

27. Michalski, R. S., "A Theory and Methodology of Learning from Examples," in *Machine Learning, An Artificial Intelligence Approach,* R. S. Michalski, J. G. Carbonell and T. M. Mitchell, eds., Tioga Press, Palo Alto, CA, 1983.

28. Minsky, M., "K-Lines: A Theory of Memory," *Cognitive Science,* Vol. 4, No. 2, 1980, pp. 117-133.

29. Mitchell, T. M., *Version Spaces: An Approach to Concept Learning,* PhD dissertation, Stanford University, December 1978.

30. Mitchell, T. M., Utgoff, P. E. and Banerji, R. B., "Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristics," in *Machine Learning, An Artificial Intelligence Approach,* R. S. Michalski, J. G. Carbonell and T. M. Mitchell, eds., Tioga Press, Palo Alto, CA, 1983.

31. Newell, A. and Simon, H. A., *Human Problem Solving,* New Jersey: Prentice-Hall, 1972.

32. Nilsson, N. J., *Principles of Artificial Intelligence,* Tioga Press, Palo Alto, CA, 1980.

33. Polya, G., *How to Solve It,* Princeton NJ: Princeton U. Press, 1945.

34. Reif, J. H. and Scherlis, W. L., "Deriving Efficient Graph Algorithms," Tech. report, Carnegie-Mellon University, Computer Science Department, 1982.

35. Sacerdoti, E. D., "Planning in a Hierarchy of Abstraction Spaces," *Artificial Intelligence,* Vol. 5, No. 2, 1974, pp. 115-135.

36. Schank, R. C., "Language and Memory," *Cognitive Science,* Vol. 4, No. 3, 1980, pp. 243-284.

37. Schank, R. C., *Dynamic Memory,* Cambridge University Press, 1982.

38. Schank, R. C., "The Current State of AI: One Man's Opinion," *AI Magazine*, Vol. IV, No. 1, 1983, pp. 1-8.

39. Shortliffe, E., *Computer Based Medical Consultations: MYCIN*, New York: Elsevier, 1976.

40. Swartout, W. and Balzer, R., "An Inevitable Intertwining of Specificaton and Implementation," *Comm. ACM*, Vol. 25, No. 7, 1982.

41. Waterman, D., Hayes-Roth, F. and Lenat, D. (eds.), *Building Expert Systems*, Addison-Wesley, 1983.

42. Wilensky, R., *Understanding Goal-Based Stories*, PhD dissertation, Yale University, Sept. 1978.

43. Wilensky, R., *Planning and Understanding*, Addison Wesley, Reading, MA, 1983.

44. Winston, P., "Learning by Creating and Justifying Transfer Frames," Tech. report AIM-520, AI Laboratory, M.I.T., January 1978.

45. Winston, P. H., "Learning and Reasoning by Analogy," *Comm. ACM*, Vol. 23, No. 12, 1979, pp. 689-703.