Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Technical
Memo
TM-91-01

Approaches to the Reuse of Plan Schemata in Planning Formalisms

Jana Köhler

January 1991

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
D-6750 Kaiserslautern, FRG
Tel.: (+49 631) 205-3211/13
Fax: (+49 631) 205-3210

Stuhlsatzenhausweg 3
D-6600 Saarbrücken 11, FRG
Tel.: (+49 681) 302-5252
Fax: (+49 681) 302-5341

Approaches to the Reuse of Plan Schemata in Planning Formalisms

Jana Köhler

DFKI-TM-91-01

Approaches to the Reuse of Plan Schemata in Planning Formalisms*

Jana Köhler
e-mail: koehler@dfki.uni-sb.de

Abstract

Planning in complex domains is normally a resource and time consuming process when it is purely based on first principles. Once a plan is generated it represents problem solving knowledge. It implicitly describes knowledge used by the planning system to achieve a given goal state from a particular initial state. In classical planning systems, this knowledge is often lost after the plan has been successfully executed. If such a planner has to solve the same problem again, it will spend the same planning effort to solve it and is not capable of "learning" from its "experience."

Therefore it seems to be useful to save generated plans for a later reuse and thus, extending the problem solving knowledge possessed by the planner. The planning knowledge can now be applied to find out whether a problem can be solved by adapting an already existing plan.

The aim of this paper is to analyze the problem of plan reuse and to describe the state of the art based on a variety of approaches which might contribute to a solution of the problem. It describes the main problems and results that could be of some relevance for the integration of plan reuse into a deductive planning formalism.

As a result, this description of the state of the art leads to a deeper insight into the complex problem of plan reuse, but also shows that the problem itself is still far from being solved.

*This research was partially sponsored by "Bundesministerium für Bildung und Wissenschaft" under grant F0 401. I would like to thank Prof. Dr. Wolfgang Wahlster for his advice and support and the members of the PHI research group, Mathias Bauer, Dr. Susanne Biundo, Dietmar Dengler, Matthias Hecking, and Gaby Merziger for their interest in my work and for many fruitful discussions. I am also indebted to the German Research Center for Artificial Intelligence from which I received additional support.

Contents

1 Introduction

Planning is concerned with the generation of sequences of actions for an agent that can change its environment. A sequence of actions constitutes a plan by which one or more explicitly stated goals can be achieved (cf. [Sha90]).

Classical AI planning systems rely upon a state-based representation of the world. These *states* are represented by sentences in a formal language. Actions that can be performed in the world under consideration are represented by *operators* that map the current state in which the action is performed into another so-called goal state.

Problems the planner has to solve are formulated by *goals* that are represented by sentences. The description of the initial state, the actions which can be performed, and the goals the planner has to solve together constitute the input. As output, the planning system produces a sequence of actions that maps the initial state into the desired goal state.

Deductive Planning uses formal logic as a basis to formalize the reasoning process. States and operators are described using a logical axiomatization. The plan is derived from the axiomatic problem description based on special kinds of theorem proving techniques. Such a deductive environment provides a well-understood means for a formal treatment of the problems which arise in planning, among them the *frame problem*, the *qualification problem*, and the *ramification problem* (cf. [MH69], [McC77], [Fin87]).

Methods of **Plan Reuse** try to extend classical planners with elements of commonsense reasoning. The main motivation is to integrate problem solving knowledge that explicitly represents former solutions to problems into the plan generation mechanism. This problem solving knowledge is encoded in plan schemata that summarize the axiomatic representation of problem solutions under a particular situation. These former problem solutions are applied when a similar problem situation occurs. They represent experience that is now useable by the planning system.

1.1 Scenarios in Plan Reuse

A definition of plan reuse should mention the various scenarios under which it can be seen. These scenarios describe different applications of plan reuse mechanisms. They vary in the role plan reuse plays when the planning abilities of a planner are considered: In the first group of scenarios, the planning abilities of the planning mechanism are extended, since it reuses plans which were developed by different planners. In the second group, the planning abilities of the system remain the same, but the performance of the planner is changed since it reuses its own plans.

- Plan reuse can be applied to extend the planning mechanism in such a way that it can deal with a larger class of planning problems. There are two scenarios in which cooperative agents work together and a reuse of plans occurs from the perspective of one planning system:
  - **Distributed Parallel Planning:** The planner belongs to a group of planners that operate on the same level of planning abilities. If a goal has to be

    achieved, it is split into various subgoals. Every planner solves one of the subgoals and generates a partial plan that achieves this subgoal. To compose the plan that meets the original goal the planners (or one supervising planner) have to consider the interactions among the various subplans. This can be viewed as a variant of plan reuse since the planners have to incorporate already existing plans into their planning knowledge.

  - **Distributed Specialized Planning:** In this scenario, a group of planners is considered in which every planner is specialized to treat problems in different domains or on different levels of abstraction. If a planner cannot achieve a goal, it communicates with the specialist planner in this field and then integrates the solution provided by this planner into its plan. This can be treated as a variant of plan reuse in which a planner has to reason about a plan that it cannot generate by itself. The scenario is also investigated in the field of blackboard architectures.

- Plan reuse can also be applied to improve the performance of a single planner and to increase the efficiency of the planning process because it can enable a planner to reflect its own reasoning process.

  - **Self-reflection:** The planner evaluates the result of its planning effort and stores plans in a knowledge base, which were evaluated as worth "remembering." If similar problems have to be solved by the system, this knowledge base is searched for a plan that can provide a solution.

  - **Hypothetical Planning:** The planner generates a plan for a future situation that is not completely described. To plan with such incomplete information the planner could add some hypotheses that enable it to generate an executable plan or it could plan on an abstract level. If at a later time the planner receives more detailed information about the problem, the hypothetical plan is reused, including necessary modification or refinements of the abstract plan.

These scenarios reflect different possible variants of the integration of plan generation and plan reuse that pose different requirements on the plan reuse mechanism.

This paper mainly concentrates on the analysis of reuse mechanisms occuring under the scenarios of self-reflection and hypothetical planning. Its aim is to describe ideas that can inspire a deductive solution of the problem of plan reuse in the framework of the PHI project at the German Research Center for Artificial Intelligence (cf. [WBH89]).

The overall objective of the PHI project is to investigate the integration of plan recognition and plan generation within help systems. Plan generation and plan recognition will be based on deductive methods. The plan generation component will contain a reuse mechanism that is also based on deductive methods. Ideas developed under the second group of scenarios are of special interest since a system consisting of one plan generation component and one plan recognition component is considered.

Some general requirements such a reuse formalism should satisfy can be formulated (cf. [Kam89a]):

- The formalism should provide a domain-independent solution to the problem of plan reuse.

- It should be integrated into the planning mechanism in such a way that it makes the planning process more efficient than planning from first principles only.

- It should work in a sound manner, i.e., if a plan is reused, the formalism guarantees that it is indeed a solution to the current planning problem.

- It should be complete, i.e., the reuse formalism is able to reuse a plan from the knowledge base of stored plans (here referred to as *plan library*) to find a solution or it requests the planner to generate a plan based on the available operators.

1.2 Phases in Plan Reuse

If plan reuse is investigated under the view of the self-reflection or the hypothetical-planning scenario, several subproblems have to be solved. These subproblems represent the different tasks a planner, with the ability of reusing previously generated plans, has to perform. These tasks are:

- Plan Determination
  The first problem a planner with an integrated reuse mechanism is faced with, is the efficient determination of a stored plan (the *reuse candidate*) from the plan library that might be appropriate to be reused. To find such an appropriate plan a formal criterion that evaluates the plans in the plan library according to the current situation is necessary. One possible criterion is to state the similarity of two planning situations and to retrieve the plan that is most similar. This variant of plan determination requires the incorporation of *analogical reasoning* into the planning formalism. Another option is to determine the plan that requires the least modification effort. The plan determination phase works similarly to a best first search: Based on the evaluation criteria, it determines the plan that best fits the current goal.

- Plan Modification
  If a plan is determined, the planner has to check whether it is a solution to the problem under consideration. This requires the evaluation of whether the plan is applicable in the current situation and whether it achieves the intended goal. This evaluation has to be performed before the plan is applied to avoid expensive modification of plans that fail. If the reuse mechanism detects potential plan failures, the plan has to be refitted. This refitting can be based on a classification of potential plan failures and corresponding refitting strategies. The current failures can be analyzed based on this classification and the corresponding refitting strategies are selected and applied to modify the plan. If plan modification fails, the plan generation mechanism has to be activated to plan from first principles.

- Plan Library Update
  A sophisticated plan reuse mechanism requires that the plan library will change

during the system's life. An advanced plan reuse component should exploit mechanisms that allow an automatic update of the plan library grounded on formal criteria that evaluate whether a plan will be added to or deleted from the plan library. Furthermore it has to be decided how plans are stored in the plan library, including the classification or generalization of plans or the creation of abstract plan schemata. This subproblem relates plan reuse to the field of *explanation-based learning* (cf. [Ham88], [M+89], [MB86]), to research considering the *update of knowledge bases* (cf. [GM88], [KM90], [Neb90]), and to the problems of *plan abstraction* and *operator abstraction* (cf. [AF88], [Kno90], [RK89], [Ten86], [UR89]). Most of the approaches to plan reuse briefly mention the problem, but do not discuss ideas towards a solution. Therefore it is also not addressed in this paper.

1.3 Overview

The aim of this paper is to analyze the problem of plan reuse and to describe the state of the art based on a variety of approaches that might contribute to a solution of the problem. It describes the major problems and results that could be of some relevance for the integration of plan reuse into a deductive planning formalism.
The approaches discussed here can be divided into two groups:

- A first group of approaches is directly devoted to the problem of plan reuse. These works are discussed here to demonstrate the various perspectives under which plan reuse can be seen. The theoretical foundations the approaches are based on are analyzed and the implemented systems they lead to are described.
- A second group of approaches is devoted to special subtasks that arise in plan reuse. Most of these works are unrelated or only loosely related to plan reuse itself. They are discussed here because they might provide a solution to one of the subtasks. We focused on approaches that addressed the problems in a logical framework. If no theoretical solution to a particular problem could be found, heuristic approaches were examined.

The paper is organized as follows:

- The second section addresses representation issues relevant to plan reuse. It briefly summarizes the requirements a representation formalism has to meet and analyzes which kinds of knowledge have to be incorporated into such a system. Then a short overview about knowledge representation formalisms used in existing plan reuse systems is given. Most of these systems use *case-based reasoning* to formalize the planning and reusing process. Therefore the basic principles underlying case-based reasoning are explained.
  In a second part of this section, we describe logical representations since we focus on a deductive treatment of plan reuse. We discuss the *situational calculus* that is widely used in deductive planning systems and review a translation of case-based reasoning into a logical formalism.

The following sections are devoted to the different subphases arising in plan reuse as described above:

- The third section describes solutions to the problem of plan determination. It summarizes methods that describe how a plan matching a given description of a planning problem can be determined. One obvious solution is to determine a plan that solves a problem most similar to the one under consideration. This similarity-based plan determination can be formalized using formal approaches to analogical reasoning. Therefore anaal reasoning is discussed in this section.
  Another approach to plan determination can be based on minimizing the planning effort. Therefore an approach to plan determination is discussed which tries to anticipate the modification effort a plan will require. The goal of the determination mechanism is to find the plan which minimizes this effort.
  Plan determination includes a second problem, which is the efficient search in a large knowledge base of stored plans. This problem is often solved based on indexing schemes. The example description of such an indexing mechanism concludes this section.

- The fourth section discusses strategies of plan modification. The section concentrates on various approaches to the analysis and classification of plan failures and discusses the heuristics that were developed to refit failed plans.
  The application of a refitting strategy often leads to the introduction of new subplans that have to be integrated into the existing plan. Therefore approaches to subgoal ordering and operator ordering problems are also discussed.

- In the fifth section, we conclude by giving an outlook on further investigations plan reuse requires. The main results are summarized leading to a possible architecture of a plan reuse system.

- In the appendix, approaches by Luria [Lur88], Hammond [Ham89] and Kambhampati [Kam89a] are described in detail since they strongly influenced the research in the field of plan reuse. They describe different implementations of plan reuse and demonstrate that plan reuse can be seen under a variety of perspectives.

2 Representation of Plans

2.1 Representational Requirements

In general, the representation of knowledge is determined by the reasoning processes that have to be performed with the knowledge that is encoded in the representation (cf. [Bra90]). Therefore the representation formalism underlying a plan reuse system has to address a number of requirements arising in plan reuse. It should cover information that is necessary to perform the integration of plan generation and plan reuse. The planning knowledge especially has to be represented so that the knowledge necessary for plan reuse can be automatically extracted from it. To provide a domain-independent solution to plan reuse inside a deductive planning formalism, it should enable the planner to automatically perform all phases of the reuse process. This requires the representation of different kinds of knowledge, i.e., of knowledge which

- informs the planner when a reuse of stored plans is useful,
- supports the computationally efficient retrieval of plans,
- helps to localize the applicability failures of a plan in a current situation,
- guides the modification of a plan if failures occur,
- supports the update of the plan library.

2.2 Relevant Approaches to the Representation Problem

2.2.1 Representation in Case-based Reasoning

Case-based reasoning is an approach to implementing dynamic memory. The basic principles underlying these ideas were introduced by Schank [Sch82] and Kolodner [Kol84]:

- Cases are usually applied to represent domain objects or other kinds of related knowledge items.
- They are organized on the basis of similarities and differences to facilitate the identification of similar cases.
- The structure of the knowledge base is dynamically updated whenever a new knowledge item is added.
- The structural changes depend on the abstraction level of the new item, how it is related to the already stored cases, and how it contradicts or fits the current structure.
- New cases are stored by relating them to the cases already stored.
- Reasoning in case bases is described as the search for a case which best fits a given description of a current problem.

Cased-based reasoning systems are usually based on some kind of *inheritance networks* that allow a hierarchical representation of the knowledge. The nodes in the nets represent the cases that are described by sets of attribute-value pairs. The links between nodes differ in the various approaches. In *discrimination nets* the links represent the discriminating features between the linked concepts, while in inheritance systems generalization or subsumption relations establish the links. The reasoning process in case-based reasoning systems is similar to the reasoning process required for plan reuse, since the basic reasoning cycle of a case-based reasoner can be described as "input a problem, find a relevant old solution, adapt it" [RS89]. Therefore it seems useful to apply some of the principles underlying case-based reasoning to plan reuse. On the other hand, the intended application is different, since case-based reasoning is normally applied in the fields of classification or diagnosis. To represent plans, the representation mechanism has to be enriched in such a way that the entities relevant to planning such as plans, actions, and goals can be represented.

2.2.2 Representation in Existing Plan Reuse Systems

- **Luria** [Lur88] stores plan schemata as unchangeable, closed units that are indexed by the goals they solve. Plan reuse is considered as an instantiation mechanism of the corresponding stored plan schemata. The modification of plans is limited to the instantiation of stored plan schemata or the replacement of predefined subplans in a more complex plan. The plan library cannot be updated. The representation is based on a KL-ONE like formalism.

- **Hammond** [Ham90] [Ham86a] develops a representation that is more detailed than Luria's approach, because he additionally indexes the plans by the failures they avoid. Causal information about potential plan failures and features of objects causing failures is attached to the plans. This information does not influence the retrieval of plans, but leads to an early anticipation of planning failures and activates the modification strategy. Information about planning failures is also used to improve the system's performance.

- **Kambhampati** [Kam89c] describes the most extensive representation of background knowledge for plan reuse. He develops a very rich representation formalism that handles the general, domain-independent structure of plans and the underlying causal relations that connect single plan steps. This representation provides the planner with a flexible strategy for plan retrieval and plan modification. The representation and organization of the plan library as a whole is not investigated in his approach since the update of a plan library is beyond the scope of the approach.

If reusing is seen as some kind of re-instantiating an old plan as by Luria [Lur88], it is sufficient to represent the applicability conditions of the plan. If the system has to be able to autonomously and reliably modify plans, additional information has to be stored. The internal structure of a plan and the decisions that occured in the planning process are important information that can be utilized by the planner. Kambhampati summarizes these requirements by stating that "one should try to reuse the planning process rather than just the result of this process" [Kam89c]. This approach to plan reuse is also denoted as "planning from second principles."

2.2.3 Macro Operators and Proof Plans

One of the earliest attempts to store and reuse generalized plans in a deductive plan generation formalism is the use of *macro operators* in STRIPS (cf. [FN71], [FHN71]). A macro operator is constructed from a plan (consisting of a sequence of operators) by replacing all constants by distinct parameters and constructing a unique list of pre- and postconditions for that operator. Macro operators are stored in *triangle tables* that provide a special triangular array structure. A macro operator can be reused if its *add-list* (representing the postconditions) partially or completely solves the planning problem. When new macro operators are added to the already stored ones, the system uses tests of redundancy and subsumption to prevent the storage of an overwhelming number of operators.

The application of macro operators in STRIPS covers the whole process of plan reuse as we described it in section 1.2, but the approach lacks flexibility since the reuse mechanism only allows the reuse of an entire macro operator or one of its components and restricts the possible modifications of reused macro operators.

Since deductive planning can be viewed as a *theorem proving* task, a number of similarities to recent research in the field of automated theorem proving are worth noting: In the deductive plan generation formalism we consider, a plan is generated by performing a constructive proof of an $\forall\exists$-formula that describes the specification of the planning problem. To prove the formula, three kinds of knowledge are necessary:

- axioms that describe the planning domain, e.g., states and basic operators,
- transformation rules to perform the proof,
- heuristics that guide the application of the transformation rules to control the proof process.

If we follow the proposal by Kambhampati to reuse the planning process rather than just its result, we have to reuse a proof process. The transformation rules and heuristics we apply to generate an applicable plan can be viewed as a proof method to solve the current planning problem.

Similar ideas are discussed in the field of automated theorem proving. Bundy [Bun90] introduces the concept of *explicit proof plans* to guide the search for a proof in automated theorem proving by reasoning about the conjectures to be proved and the methods available to prove them. Proof plans are specifications of the underlying proof strategies. They have the following properties:

- **Usefulness:** The proof plan reduces the search effort.
- **Generality:** It succeeds in a large number of cases.
- **Expectancy:** A proof plan additionally contains some knowledge that can be used to predict when it will succeed or fail.
- **Uncertainty:** Proof plans are used in undecidable domains and it is not possible to always correctly predict if the proof plan will succeed or fail.
- **Patchability:** Failing proof plans can be repaired by providing alternative steps.
- **Learnability:** New proof plans can be automatically learned.

Research in the use of explicit proof plans was motivated by a project to develop automatic search control for the NuPRL program synthesis system [C$^+$86], which proves theorems by mathematical induction. To achieve this, the Boyer-Moore theorem prover [BM79] with its use of heuristics to guide inductive proofs is rationally reconstructed and a formal account of the heuristics is developed. This formal account includes the ability to predict the circumstances under which a heuristic will succeed or fail. Heuristics will be applied in a flexible way. The intention is to learn new heuristics from example proofs. Proof plans are recursive and contain subroutines.

To find an abstract proof plan Kerber [Ker88] proposes the following way to classify theorems and then to look for an analogy:

1. find the corresponding category of the problem,

2. find an "analogous" theorem in this category with a known proof,

3. try to transform the proof so that it is applicable to prove the new theorem.

Analogies can be exploited on two levels. On the level of singular proof steps they are helpful to reduce the search space for a proof by splitting up a theorem into lemmata and reusing already proved lemmata regarding the proofs of lemmata as "macro" steps. On the level of abstract proof plans analogies are used to transform machine generated proofs from a logical level into a conceptual level to explicate the inherent structure of the proof. Brocks et al. [BCP90] investigate how analogies can be applied on the level of single proof steps rather than whole proof plans and discuss modification strategies for mathematical proofs. Lingenfelder [Lin89] discusses the structuring of proofs to improve the presentation of a computer generated proof to a human user.

Proof plans can be a priori encoded in a theorem prover (cf. [Bun90]) or they can be learned by the system based on an analysis of proof processes (cf. [M^{+}89], [MB86]).

2.3 Logical Representations

The integration of plan reuse into a deductive plan generation formalism requires the investigation of how plan reuse formalisms can be described using deductive representation formalisms. The plan generation formalism in which we intend to integrate a plan reuse facility will be based on a method of deductive program synthesis [Biu90]. As logical representation formalism, *situational calculus* has been applied and therefore we briefly describe this most widely used logical representation.

2.3.1 Situational Calculus

Situational calculus was developed by McCarthy and Hayes [MH69] to provide a language that can be used to formalize reasoning about actions. It provides expressions in which situations and actions can be described.

A *situation* s is defined as the complete state of the universe at an instant of time. Because of their complexity, situations cannot be completely described but are represented by facts about situations. To allow a situation-dependent interpretation of terms, the notion of a *fluent* is introduced.

A *fluent* f is defined as a function whose domain is the set of all situations. If the range of the function is represented by the truth values $\{true, false\}$, it is referred to as *propositional fluent.* If the range is the set of all situations, it is referred to as *situational fluent.* Important situational fluents are $next(\pi)$, which denotes the next situation s' in the future of a situation s in which $\pi(s')$ holds, and $result(p, \sigma, s)$ (where p is an agent, σ is an action, and s is a situation) is the situation that results when p carries out σ, starting in the situation s. To express that a relation holds in a situation, an extra state parameter is added to that relation.

A third important element of the representation (beside situations and actions) is assertions about *causality*, which are represented by fluents of the form $F(\pi, s)$, where π is itself a propositional fluent and the expression asserts that the situation s will be followed (after an unspecified amount of time) by a situation s' in which the fluent π holds.

This formulation of the situational calculus by McCarthy and Hayes provides the basic elements to formalize planning knowledge and allows a formal treatment of important problems which arise in the area of planning, e.g., the problem of *temporal projection* [HM86]. Temporal projection can be described as the form of reasoning in which the properties of the world in a resulting situation are predicted if some description of a current situation, of the effects of possible actions, and of a sequence of performed actions is given. It is an important problem in the field of planning, since verifying that a plan achieves a given goal is a temporal projection problem. Two important problems arise during temporal projection: the frame problem, which requires the determination of all the properties of a situation which are not affected by an action, and the qualification problem, which expresses that the successful performance of an action may depend on a large number of qualifications that also have to be considered.

To generate a plan that is guaranteed to be executable, Manna and Waldinger (cf. [MW86], [MW87]) propose an approach to deductive planning were proofs have to be *constructive*, in the sense that they only contain plan fluents referring to the initial state and do not rely on arbitrary hypothetical states which might not be executable.

Further approaches provide a solution to the temporal projection problem based on non-monotonic formalisms (cf. [Sho86], [Kau86], [Lif87], [LR89], [Bak89], [Luk88], [GS88a], [GS88b], [BH90]).

2.3.2 A Translation of a Case-based Reasoning System into Default Logic

The representation formalism to be developed for a plan reuse system should cover two aspects. The plans to be used are represented in a logical framework the deductive plan generation formalism provides. These plans have to be stored in the plan library for which hierarchical knowledge representation formalisms such as, e.g., inheritance networks are widely used. An interesting question is now, whether a uniform representation can be developed in which plans and also the plan library can be represented. To find an answer to this question, an analysis of how the various representation formalisms can be translated into each other is helpful. The following approach is an example of such a translation of case-based reasoning into a logical formalism.

Koton and Chase [KC89] describe how case-based reasoning can be formalized using default logic [Rei80]. They develop a translation to transform a graph-oriented representation into default theories. To find a similar case, the system constructs extensions that represent consistent sets of beliefs due to the current situation description and the information in the case base. Etherington [Eth88] describes how inheritance systems can be reformulated using semi-normal default theories. Koton and Chase demonstrate in their approach that for a special kind of inheritance networks, the so-called discrimination networks, normal default theories are sufficient to formalize the inference mechanism.

They use the representation proposed by Kolodner [Kol84]. The case base is represented using a discrimination network. Cases are described by so-called *features* that represent attribute-value pairs. Similar cases are grouped into *gens* which are generalization structures created by the system. Gens contain the features that are common to most (here interpreted as 2/3) of the cases, which are called the *norms*. The cases belonging to the same gen are indexed by the features that distinguish them from each other.

The nodes represent individual cases or gens and are related by features. A subnode is linked to its parentnode by the features that differentiate the subnode from the parent gen. The set of these links represent the *diffs* of the gen. The indexes are represented using two levels, the *category* and the *value* of the feature, for example `covering=feathers` where `covering` is a category and `feathers` is a value. A set of features defines a path in the discrimation net used to search for a similar case. These concepts described above are now translated into defaults [KC89]:

1. **Feature:** A node $N(x)$ has the feature $f(x,a)$ and there are no diffs at this node corresponding to this feature, i.e., the subnodes of this gen normally have the same feature as the gen.

$$\frac{N(x) : f(x,a)}{f(x,a)}$$

2. **Link:** A node $N(x)$ is linked to a gen $G(x)$ by a diff $f(x,b)$, i.e., the node $N(x)$ is a subnode of $G(x)$ and possesses the different value b for f.

$$\frac{N(x) \wedge f(x,b) : N(x)}{N(x)}$$

3. **Differentiated Norm:** $G(x)$ is a gen and has the norm $f(x,a)$ and the diffs $f(x,b_1), ..., f(x,b_n)$, meaning that the subnodes belonging to the gen usually have the value a for feature f, unless there are exceptions.

$$\frac{G(x) \wedge \neg f(x,b_1) \wedge ... \wedge \neg f(x,b_n) : f(x,a)}{f(x,a)}$$

4. **Restricted Closed World:** A node $N(x)$ is linked to a gen $G(x)$ with norm $f(x,a)$ by the diffs $f(x,b), f_1(x,b_1), ..., f_n(x,b_n)$, i.e., if $G(x)$ cannot be specialized to $N(x)$ by these diffs, then $G(x)$ has not value b for feature f.

$$\frac{G(x) \wedge \neg f_1(x,b_1) \wedge ... \wedge \neg f_n(x,b_n) : \neg f(x,b)}{\neg f(x,b)}$$

The set of axioms contains the formula $\forall x : G_o(x)$ stating that all cases belong to the root gen that represents the generalization of all cases stored in the case base. For every new case that is stored in the case base, the four default rules as described above have to be added to the default theory to represent the relationship between a gen and one of its specializations.

The inference mechanism uses the defaults in the following way: To the set of axioms first order assertions are added that describe the features of the current case. These assertions define the search path in the case base since the inference mechanism follows the generalizations of the features matching the assertions. The norms that can be collected when following the path represent the knowledge that can be inferred for the case. This process corresponds to the construction of an extension in the default theory.

The application of normal default theories to reformulate case-based reasoning guarantees two properties:

1. Normal default theories have at least one extension that assures that for every case a consistent set of features can be derived.

2. Normal default theories are semi-monotonic, guaranteeing that the addition of new cases to the case base (in the form of new defaults) leads to an extension that contains the former extension as a subset.

If multiple extensions can be constructed (by following different paths in the discrimination network), extra-logical inference rules have to be applied that select one of the extensions to be preferred. The order in which the defaults are applied influences the order in which the extensions are constructed.

3 Plan Determination

In section 1.2 we described plan determination as the first task a planner with an integrated reuse mechanism has to perform, when a planning task has to be solved by reusing a stored plan. Carrying out this task involves a *comparison* of the current problem description presented to the system with stored problem descriptions the system has already dealt with. This is done by what we call an *analogy-driven* approach to plan determination because the plan that is most analogous to a given problem description has to be selected as a reuse candidate. The comparison can be seen as an *interpretation* of a candidate plan in a new situation in which a mapping between the new goal and the retrieved plan is developed. Such a process of interpretation is similar to problems studied in the field of analogical reasoning. Thus we provide a short introduction into analogical reasoning and investigate whether results from this field could be helpful in formalizing the process of comparison underlying plan determination. We briefly review two approaches that try to formalize analogical reasoning in a deductive formalism and in which analogy is not reduced to partial or complete syntactical identity.

In a second subsection, another view of plan determination is investigated. It is referred to as the *efficiency-driven* approach in which the effort that is necessary to modify a plan is used to guide the search for a reuse candidate. From the knowledge base the plan requiring the least modification effort is selected.

Besides the determination of a reuse candidate based on a current problem description, a second problem that has to be solved is the overall organization of a large base of stored plans and the efficient search in it. A solution might incorporate existing retrieval mechanisms from case-based reasoning systems that especially investigate the problem of how to search in a very large memory of stored cases. The retrieval mechanisms perform a straight-forward feature matching to find a similar case in the case base. While the approaches to analogical reasoning as described here mainly concentrate on the development of a sophisticated matching procedure based on heuristic knowledge, the retrieval mechanisms which were developed for case-based reasoning systems often use a simplified matching procedure but concentrate on the development of a sophisticated indexing scheme as a basis for an efficient search and retrieval mechanism. An elaborate plan determination procedure should incorporate both.

3.1 Analogy-driven Plan Determination

The formalization of analogical reasoning is still an open problem. Most of the approaches try to formalize analogical reasoning based on other kinds of reasoning formalisms, as for instance, on inductive inference [Tho88], on an extension of first order logic [DR87], [Gre88], [Mun81], and on statistical reasoning [Lou89]) or on the use of set theoretic models, fuzzy set approaches or semantic and model-theoretic approaches (cf. [Thi86], [HA96]).

We will discuss approaches to analogical reasoning that are related to deductive reasoning. At first we define a concept of analogy based on the approaches by Gentner [Gen88], Indurkhya [Ind88] and Hall [Hal89]. Then the work by Greiner [Gre88] and Munyer [Mun81] is discussed.

3.1.1 A Concept of Analogy

In all computational approaches considered in this paper, analogy is described as a *mapping* between elements of a *source* domain and a *target* domain. In our application, the plans of the plan library correspond to the source and the current goal description is an element of the target. The analogy maps elements from the source into the target domain. The mapped elements represent the analogical inferences the system draws. They are justified by the initial elements of the mapping.

One concept of analogy that can be usefully applied in our context is characterized by Indurkhya [Ind88] as *predictive analogy*, which refers to the process of justifiably inferring further similarities among objects or situations based on some existing similarities. The inferred similarities form the basis of a prediction about the target from the known features of the source.

This kind of analogical reasoning can be further analyzed applying the *four-component process model* of analogy developed by Hall [Hal89], as the basis of a comparative analysis about computational approaches to analogy. In this model four separate phases constituting the analogical reasoning process are distinguished (cf. figure 1):

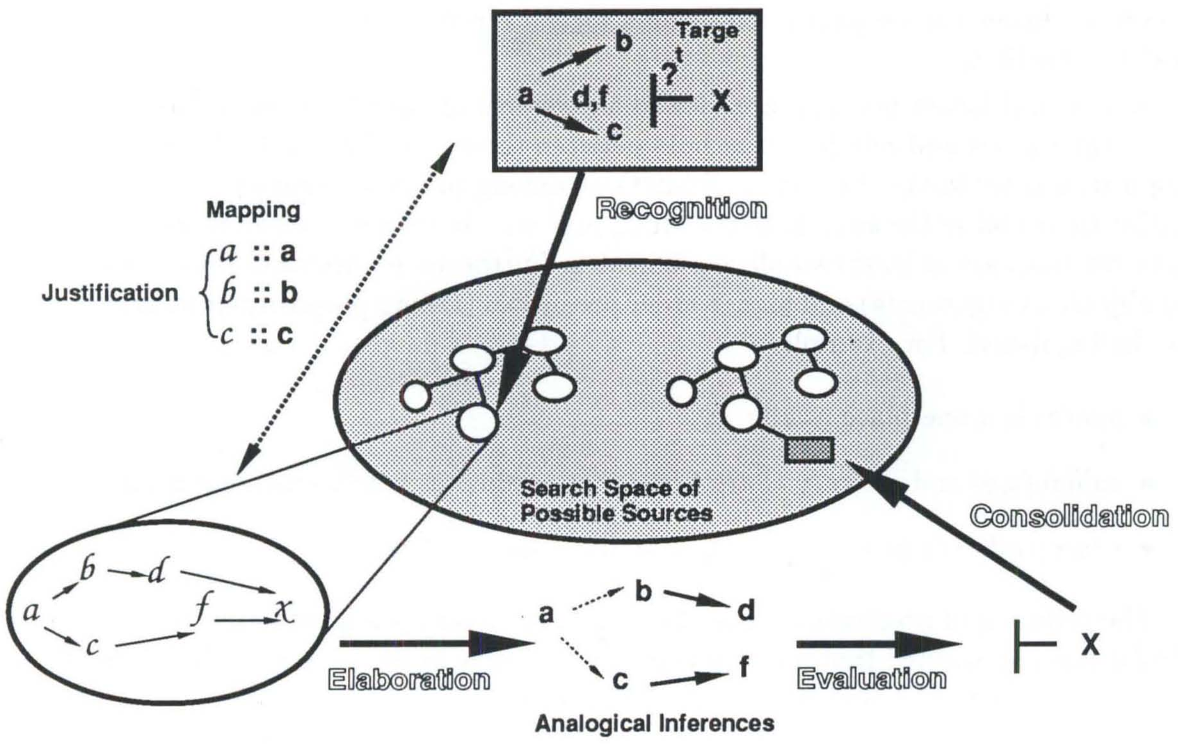

Figure 1: An Illustration of Analogical Reasoning

Recognition describes the search process for an analogous candidate source given a target description. The central problem is to restrict the retrieval process in the set of possible sources.

Elaboration draws the analogical inferences after a source domain is selected and the basic mappings between objects are established. These basic mappings are the kernel of the *justification* or *set of support* for the analogy.

Evaluation examines the soundness of an analogical mapping. Furthermore it applies the mapping and inferences in some context of use and leads to the solution of the problem in the target domain including further justification of the analogy, repair of the mapping when incorrect results occur, or extension of the mapping if the analogy successfully solves problems in the target domain.

Consolidation stores the confirmed inferences, so that the results of the analogy can be usefully reinstantiated in other contexts as well.

A computational model of analogical reasoning should cover all four phases.

3.1.2 The Structure Mapping Model

The *structure-mapping model* developed by Gentner [Gen88] is one of the basic approaches to the computational description of analogy. It describes some principle ideas of how analogies can be discovered using a logical representation and exploiting syntactical information.

Source and target domains are viewed as systems of objects. Objects are characterized by attributes and relations that hold between them. The knowledge is represented in propositional networks. Syntactic distinctions among predicate types provide the information to formalize the analogical reasoning process: Attributes are one-place predicates, while relations are at least two-place predicates. Furthermore, first-order predicates (taking objects as arguments) and higher-order predicates (taking propositions as arguments) are distinguished. For example:

- *sun(x)* is a one-place attribute
- *collide(x,y)* and *strike(y,z)* are first-order predicates which represent relations
- *cause(collide(x,y),strike(y,z))* is a higher-order predicate

The ordering of predicates is based on the basic assumption that only those systems of relations are mapped that are connected by causality in the source. This is formulated as the so-called *Systematicity Principle* (cf. [Gen88]):

> "A predicate that belongs to a mappable system of mutually interconnecting relationships is more likely to be imported into the target than an isolated predicate."

Analogies are characterized as one out of four different kinds of possible comparisons between *source* and *target* based on the proportion of attributes and relations that can be successfully mapped. Gentner illustrates this classification by the following examples:

Literal Similarity: Most of the object-attributes and the relational predicates can be mapped.

"The k5 solar system is like our solar system."

Analogy: Most of the relational predicates, but few or no object attributes, can be mapped as in the famous analogy of the atom model by N. Bohr.

"The atom is like our solar system."

Abstraction: The source is an abstract relational structure (a theoretical concept); all abstract predicates are mapped into the target domain.

"The atom is a central force system."

Anomaly: Only a few or no attributes and relations can be mapped.

"Coffee is like the solar system."

The syntactic type of the shared versus nonshared predicates determines whether a given comparison is thought of as analogy, as literal similarity, as anomaly, or as an abstraction. Analogies are constructed by mapping the objects of the target onto corresponding objects in the source. Then the attributes of the source objects are discarded because they are not mapped into the target domain. In a last step the existing relations between the objects of the source have to be mapped to infer relations that hold in the target. In this step the information about the order of predicates is exploited: Higher-order predicates enforce connections among lower-order predicates and support the choice of the right number of relations that construct the analogical mapping.

Gentner mainly concentrates on the problem of which object attributes and relations will be mapped in constructing an analogy. As a fundamental result, he proposes to concentrate on systems of relations when a mapping has to be constructed. This result is formulated in the systematicity principle. An open problem is the construction of the correct object mapping that requires a solution of the above mentioned recognition problem.

3.1.3 Logical Approaches to the Formalization of Analogical Reasoning

We describe two approaches that formalize analogical reasoning by extending deductive inference systems.

Introducing an Analogical Inference Operator

Greiner [Gre88] defines an *analogical inference operator* $|\!\sim$ to formalize analogical reasoning within a deductive framework. Analogies are constructed as instantiations of stored *abstractions.* Abstractions represent solution methods to past problems in the domain under consideration. They are stored as formula schemata and defined such that problem solutions belonging to different domains are connected via the same abstraction. To find the correct abstraction, the system is provided with an explicit hint towards the underlying object correspondence between source and target it has to consider. If an analogy has to be constructed, an abstraction is found by which it is connected to the source object. It is then instantiated in the target domain leading to a new set of formulas that provide additional information about the target object. To solve the problem the original theory is augmented with this formula set. To construct an analogy the system has to be provided with:

1. axioms in a theory Th for target and source domains,
2. a problem statement PT in the target that has to be solved,
3. a hint that the target concept A is like source concept B, A $\sim$ B.

It searches for a conjecture about a target concept $\varphi(A)$ as an instantiation of an existing abstraction, i.e., , Th $\models AbstForm(\varphi(A))$ and Th, A $\sim B \;|\!\sim_{PT} \varphi(A)$ hold.

The derived formula has to satisfy a number of conditions to be accepted as analogical hypothesis:

1. The target conjecture does not hold in Th.
   $Th \not\models \varphi(A)$
2. The target conjecture leads to a consistent extension of Th.
   $Th \not\models \neg\varphi(A)$

3. A source instantiation of the considered abstraction holds in Th.
   $Th \models \varphi(B)$

4. When the theory TH is extended consistently, PT follows.
   $Th \cup \{\varphi(A)\} \models PT$

The system uses an iterative generate-and-test control structure: The generator finds an abstraction for source and target based on the information in the hint provided by the user. To reduce the search effort, heuristics are applied, which help in choosing an abstraction among the candidates. A test process determines whether an abstraction can be used to solve the target problem proving the conditions defined above. These proofs are only partially mechanized, e.g., to prove the correctness of φ, the user is asked to approve the conjecture. An additional lack of automation is the necessity of providing the system with an explicit hint about the object correspondence between source and target. As in the theory developed by Gentner, no complete solution to the recognition problem is obtained.

Extending the Underlying Unification Mechanism

Another way of formalizing analogical reasoning is to extend first-order-unification with an analogical mapping algorithm as proposed by Munyer [Mun81]. He applies analogical reasoning in performing equality proofs of mathematical expressions by doing equivalent transformations. The proposed unification mechanism contains:

- many-to-one bindings,
- commutative or associative reorderings among arguments of predicates, and
- erasure of terms, e.g., functions of different arities can be unified by dropping some of the arguments.

As an example the problem of unifiying two terms with different leading function symbols, e.g., $f(a, g(b))$ and $h(g(c), a)$ is considered. These terms can be unified if the underlying theory, for example, provides the commutativity of f together with an equation $f(x, y) = h(x, y)$ for all x and y.

An analogy is found if the candidate source derivation contains an operator sequence which terminates in a source formula that consistently maps (using the extended unification) to a known goal formula in the target search space.

This extension of unification is similar to *theory unification* ($\mathcal{T}$-unification), which was originally developed by Plotkin (1972). $\mathcal{T}$ is an axiomatization of an equational theory involving some of the function symbols. $\mathcal{T}$-unification can be viewed as solving equations in an underlying theory: Given an equational theory and a pair of terms, substitutions are computed for the variables, such that both terms become equal under the equational theory (cf. [Sie89] and [Bür86]).

An application of theory unification is first of all useful in formalizing the recognition phase in the analogical reasoning process by exploiting knowledge about the equality of objects and relations holding between them. Nevertheless theory unification incorporates a number of theoretical and practical problems: A unique, most general unifier is no longer guaranteed to exist (there might be infinitely many most general unifiers or none at all) leading to the non-termination of the unification procedure. But even if finitely many most general unifiers exist, the unification procedure becomes very expensive (cf. [Sie89]).

3.1.4 Implementations of Analogical Reasoning

Falkenheiner and Forbus [FF86] describe an implementation of Gentner's *structure-mapping theory* [Gen88]. The *structure-mapping engine* SME is a cognitive simulation program to study human analogical behavior and provides a tool-kit with which matching algorithms based on Gentner's *structure-mapping theory* [Gen88] can be developed. SME uses only the domain information provided for the target in constructing all syntactically consistent analogical mappings between source and target. To construct possible matches between elements of source and target, so-called *match hypotheses constructor rules* are used. These rules compare syntactical elements of the facts that represent source and target and lead to a number of *match hypotheses.* The individual match hypotheses are evaluated, applying *match evidence rules* to the hypotheses. These rules add a probability factor to every match using the formalism developed by Dempster and Shafer (cf. [Sha76], [Dem67]). From all possible syntactical matches, the maximal sets of consistent match hypotheses are collected to construct possible analogical mappings. To find the best analogical mapping, a number of heuristics are applied. They include the evidence factors for the individual match hypotheses, the possible analogical inferences that could be drawn, and the graph-theoretic structure of the potential analogical mapping, representing the number and relative size of the connected components.

Leishman [Lei89] describes the implementation of an analogical reasoner using a graph-oriented representation. The reasoner searches for *minimal common generalizations* of subgraphs to establish a correspondence between these subgraphs to construct an analogical mapping of the elements in the subgraphs.

As an example of how analogies can be applied in plan reuse, Daube and Hayes-Roth [DHR89] describe their practical experience with the implementation of analogical reasoning applied to design problems. Design is treated like a planning task, where a new design problem is solved retrieving a stored design plan.

3.2 Efficiency-driven Plan Determination

Now we describe the approach of Kambhampati (cf. [Kam89c], [Kam89b], [Kam90b], [Kam90a]), who uses the costs a plan modification requires as a criterion for the determination of a reuse candidate. In the system PRIAR the integration of classical hierarchical planning with plan reuse is investigated. A basic assumption is that the planner exploits its reuse facilities only if they lead to an improved performance of the planning process. To maximize the efficiency of the planning process three critical cost factors have to be considered:

1. **Retrieval** costs to determine a plan from the plan library.

2. **Mapping** costs to interprete the retrieved plan in the new planning situation.
3. **Modification** costs to refit the retrieved plan so that it can be applied to the new planning situation.

The minimization of these cost factors is necessary if plan reuse should usefully be integrated into a planning system. Following this idea, Kambhampati proposes the development of a retrieval mechanism that considers the costs of a necessary modification and chooses the plan for reuse that demands the least modification costs [Kam90a]. He proposes a domain-independent solution that does not depend on any prior knowledge of the solution for the new problem, but instead makes an informed estimate of the modification costs a candidate plan would require being applied in a new planning situation. The retrieval process is restricted to the selection of a plan from a set of potentially applicable plans. To select this set, a partial unification [KH89] of the new goal description and the goals the stored plans solve is applied. The retrieval mechanism is described in more detail in the appendix.

A similar approach to *validated retrieval* is described by Simoudis and Miller in [SM90]. This work is strongly application oriented and does not introduce any new theoretical ideas but supports the idea of validations used in retrieval with some experimental results.

3.3 Search Procedures to Handle Large Knowledge Bases

As an example, we briefly describe the main ideas of the indexing scheme underlying the graph search procedures developed by Stottler, Henke and King [SHK89]. They developed an algorithm for retrieval in a case-based system that can handle very large knowledge bases in which up to millions of cases are stored. In their approach, they distinguish between *quantitative* and *qualitative* retrieval and a mixed form of both.

Quantitative retrieval is applied in domains where all attributes can be described by numeric values and where all attributes are of the same importance when characterizing the case. The similarity of two cases is defined as the inverse of their distance from each other in the case base. The distance is measured as the geometric distance between two points in an n dimensional hypercube where n is the number of attributes. The algorithm divides the hypercube into subcubes. It searches for the subcube that contains the current case, represented by a point in the n dimensional space spanned by the attributes. The algorithm halts when the subcube contains no more than a predefined number of points. The points that are in the same subcube as the current case represent the most similar cases that can be retrieved for that case. One disadvantage of this algorithm is that it uses a static similarity metric that treats similarity identically throughout the case base, while in real applications the similarity measure among cases might change.

Qualitative retrieval handles domains in which attributes are described qualitatively, using predefined concepts of the domain. Similarity is defined here as the number of syntactically equal attribute values that can be augmented by a weighting scheme to represent the varying influence of different attributes. Every case is indexed by all possible combinations of attributes that are used to describe the case. For example, if a case has attributes a_1 and a_2, it is indexed by $[a_1, a_2, a_1a_2]$. These indexes describe the multiple pointers a tree-hash algorithm maintains to the cases. The performance of the algorithm is independent of the number of cases but slows down when the number of attributes increases.

Further experiences with case-based search algorithms are described by Bradtke and Lehnert [BL88].

4 Plan Modification

After the plan determination phase, a candidate plan is found for reuse. We assume that exactly one candidate plan will be considered. This plan will be investigated whether it is applicable in the current planning situation or not. "To be applicable" is used here in the sense that the preconditions the plan requires hold in the current state of the world and, of course, the plan solves the current goal. The plan modification mechanism has to perform two tasks:

1. In a first step, it has to determine the parts of the plan that might fail and the subgoals that remain unsolved in the current planning situation.

2. In a second step, it has to determine appropriate modification strategies to refit plan failures.

It should be mentioned that approaches exist that are able to avoid the modification phase. In these approaches, the new plan is made up of several candidate plans from which only the applicable parts are taken. An example of this approach is described by Redmond [Red90]. He uses case pieces, so-called *snippets*, to construct a new case from multiple stored cases. Furthermore, plan modification should be seen in contrast to plan debugging. In plan debugging, the candidate plan is immediately applied in the current planning situation (or simulated in a world model) after it has been determined. If it fails, a debugging strategy is activated based on information taken from the plan application. Most of the plan debugging techniques (cf. [Ham86b], [Chi89], [Sim88]) analyze the application failures and try to develop causal explanations for why these failures occured. Based on these explanations the plan is refitted. The aim of plan modification, as considered here, is to recognize and refit plan failures before the plan is applied. The plan modification mechanism has to recognize potential plan failures and refit them before the plan is reused. This process is more complicated than plan debugging because no failure information is directly available.

4.1 Plan Failures and Refitting Strategies

An obvious solution to the plan modification problem is to classify potential plan failures and to develop a domain-independent refitting strategy for each class of failures. Therefore we review two approaches which describe the first steps towards this solution and which mainly influenced the field, even though the problem is addressed in an informal way. Then we describe the first formal treatment of plan modification that additionally includes a mechanism for recognizing potential plan failures without any simulation or debugging mechanism.

Another problem that has to be addressed in the plan modification phase is the ordering of operators and subgoals, since the order in which the subgoals are solved mainly influences whether a plan can be found. We review approaches which try to recognize operator and goal interactions to control the plan generation process.

4.1.1 Failure Classification Based Approaches

Wilensky [Wil83] is one of the first to address the problem of plan failures. It is investigated in the context of language understanding, where reasoning about goals is necessary for the recognition of intentions. People usually have a number of intentions and thus the problem of interacting goals and plans has to be investigated. Wilensky introduces a classification of so-called *negative goal relationships* which could form the basis for a classification of plan failures. Three kinds of negative goal relationships are distinguished:

1. **Resource Limitations:**
    - *Time conflicts* occur when a deadline has to be met to successfully reach a goal or when actions have to be synchronized.
    - *Consumable functional objects* represent a limitation when an action requires more of them than are available in the current world state.
    - *Nonconsumable functional objects* are not reduced when an action is performed (in contrast to consumable functional objects), but set a limitation because of their limited capacity.
    - *Abilities* refer to the limitations the person for whom a plan is generated sets on the plan.
2. **Mutually Exclusive States:**
    - The subgoals themselves lead to inherently contradicting world states.
    - A consequence of an action excludes a subsequent action that is necessary to achieve the goal.
    - A precondition for an action excludes the achievement of one of the subgoals.
3. **Causing a Preservation Goal:**
    - This refers to the problem where a plan solves the intended goal, but additionally has a side-effect that leads to an undesired state or destroys a state which should persist.

This classification is inspired by the application Wilensky investigates and is not immediately applicable to support plan reuse. But it might inspire a deeper analysis of precondition failures. One failure that is not addressed is the possibility that operators are combined such that their common effect prevents the execution of another operator while the single operators do not affect it.

To solve the goal conflicts, Wilensky proposes a meta planning heuristic that influences the way in which the conflicts are treated, leading to three different conflict resolution strategies:

1. **Goal Conflict Resolution:**
   The conflict between the goals can be solved and a plan is generated that achieves both.

2. **Goal Abandonment:**
   The conflict cannot be solved and one of the goals has to be abandoned to find a plan that achieves the remaining goal.

3. **Spontaneous Goal Conflict Resolution:**
   The goal conflict is solved without any activities by the planner because of some external event that leads to a change in the world state.

The last two strategies seem a bit strange when considering classical planning problems, but might be quite appropriate in real world planning or everyday planning, which is investigated by Wilensky. The strategies themselves are described very informally, e.g., if conflict resolution is selected as a meta-goal, the following ordered strategies are applicable:

1. For every conflict a plan is stored in the library of plans that solves the conflict. This subplan is included into the original plan before the conflict occurs.

2. The subplan that causes the conflict is exchanged by another applicable plan that solves the same subgoal.

3. Before the plan is performed, the world state is changed to avoid the goal conflict, i.e., a subplan is added in front of the generated plan.

Wilensky's work is of historical interest. The approach lacks a formalization and the problem of how to recognize potential plan failures is not addressed.

Another approach that inspired research in plan reuse was developed by Alterman [Alt86] who introduced *planning by situation matching.* The planning system extensively exploits different kinds of background knowledge like abstract plans, categorization and causal knowledge to compare old and new planning situations. Plans are represented in an abstraction hierarchy containing specific and general plans ordered by the purpose (goal) they are used for. After the comparison of the two planning situations, four kinds of situation differences can be distinguished:

- A *failing precondition* forces the planner to move the abstraction hierarchy up along the purpose-relation for the plan and to look for a more abstract plan that does not require the failing precondition. Another instance of this abstract plan is used to try to resolve the conflict.

- The *step-out-of-order* conflict is solved by reordering the steps of the plan or by removing an intermediate step. The reordering is determined by again searching through the abstraction and specialization hierarchy until a plan is found that contains the right order of planning steps.
- If *differing goals* occur, the plan that is to be reused solves more than the intended goal. This is not treated as a conflict, since the plan additionally solves the goals which had to be solved.
- A *failing outcome* shows that one of the expected goals does not hold after the plan application. The conflict is resolved by determining the missing planning steps and adding them to the plan.

Altermans idea of adaptive planning is limited to the selection of alternative plans or subplans from a predefined plan hierarchy. This does not represent a modification in the sense considered here.

Hammond [Ham90] describes a very detailed and comprehensive collection of conflict strategies treating five categories of conflicts which are similar to the classifications described above. Each conflict category is connected to a global refitting strategy, which is again refined by various refitting methods. It is the recent work on this topic and could, perhaps, be taken as a basis to develop a formal treatment of the modification problem. It is not described in detail here, because the basic idea is the same as in the two approaches described above.

4.1.2 A Theory of Plan Modification

The first formal treatment of plan modification was described by Kambhampati in [Kam90b], who tried to formalize methods showing how conflicts in reuse candidates can be recognized, and strategies to resolve these conflicts. The application of these strategies and the modification of the plan is left to the planner, whereas the reuse component detects the failures and recommends the strategies with which the planner can overcome the failures. The plan modification mechanism is based on the representation formalism as described in section C.1.

Applicability failures, redundancies and shortcomings are formally characterized as inconsistencies of the *validation structure.* Therefore the validation structure of a plan that is a potential candidate for plan reuse is checked for consistency in the new planning situation. Inconsistencies in the validation structure inform the planner that the plan is not reuseable and has to be modified. This modification process changes the plan in such a way, that the inconsistencies in the validation structure are removed. A consistent validation structure is by construction sufficient for a plan to be successfully executable, since it reflects the intermediate planning decisions and dependencies between the planning steps.

The modification process is divided into two steps:

1. The first step (annotation verification) is the consistency check of the validation structure. The analysis of the validation information allows the determination of

potential plan failures if the plan were to be applied. These potential failures are reflected by inconsistencies in the validation structure.

2. Every recognized inconsistency is classified according to the types of failures known by the system. To remove the inconsistencies, the system uses so-called *refit tasks.* The plan and the refit tasks are sent back to the planner, which performs the necessary modification of the plan in the second step (refitting).

The modification algorithm guarantees the correctness of the modification process, but not its optimality. It is described in more detail in the appendix.

4.2 Goal and Operator Ordering Strategies

Goal ordering problems arise when the overall goal for the planner is given as a conjunction of subgoals. This conjunction has to be split up into the various subgoals that are separately addressed by the planner. The order in which these subgoals are solved mainly influences the complexity of the planning process.

Operator ordering problems occur when the same goal can be achieved by different operators representing different actions. The planner has to decide which operator to select.

The application of ordering strategies in planning is discussed under two views:

- **Conflict solving** among interacting subgoals is necessary in every planning mechanism that works beyond simple toy domains.
- The generation of optimal plans (where optimality can be defined, for example, as the minimal number of planning steps to solve a problem) demands the consideration of **efficient planning** strategies that are able to detect commonalities among operators.

Hertzberg and Horz [HH89] and Cheng and Irani [CI89] describe formal approaches to the problem of subgoal ordering that are applicable for STRIPS-like planners. These planning mechanisms are based on two assumptions:

1. The *STRIPS-assumption* requires that domain conditions only change if this is explicitly stated by the postconditions of operators and that operators are defined such that their application unambiguously maps situations onto situations.

2. The *locality-assumption* requires that every operator specifies all domain conditions it changes regardless of the situation in which it is applied.

Both assumptions restrict the representation of plans and operators such that it is completely decideable if an operator can be applied. Hertzberg and Horz exploit the predefined partial ordering about operators to detect conflicts in nonlinear plans, while Cheng and Irani compute an ordering relation between sets of subgoals based on the operators that can be applied to solve these subgoals.

Drummond [DC89] informally discusses a heuristic approach to the subgoal ordering problem. He develops a so-called *temporal coherence* heuristic that reduces the search space for the planner.

Hayes [Hay89] describes an approach to increase the efficiency of the planning process using knowledge about overlapping subgoals and operators and informally discusses the advantages of exploiting operator overlap using an example from the mechanics domain. The task is to start with a rectangular block of metal and to generate a plan for a CNC machine to cut a variety of shapes or holes into the block. She uses positive interactions among operators to make plans more efficient by exploiting temporal dependencies among the operators. The idea is to shorten the plan by grouping the subgoals that overlap together and by choosing for each subgoal the operator which overlaps most of the operators already contained by the plan.

5 Conclusion and Outlook

5.1 Summary of Results

With this paper two goals were pursued:

1. A comparison of existing approaches to the reuse of plan schemata to describe the state of the art for this problem and to develop a model on which further research can be based on.

2. The analysis of various subproblems that are connected with plan reuse and a discussion of results from other research areas which might contribute to a solution.

The following results have been obtained:

- The investigated approaches to plan reuse demonstrate the different views of the problem in general. Until now there exists no approach aimed at an integration of plan reuse and plan generation on a deductive basis. The PRIAR system (appendix C) is the one which uses an approach comparable with the one we intend. CHEF (appendix B) and KIP (appendix A) are pure reuse systems and do not have a plan generation facility. None of the investigated systems provides a formalization of the reuse process based on logic.

- Existing plan reuse systems adapt principles from case-based reasoning in their representation formalism [Lur88], [Ham90], or develop their own representations that are especially devoted to plan reuse [Kam89a]. Pure case-based reasoning systems are only able to plan by reusing plans which are stored in the plan library. They fail if completely new goals have to be solved for which no applicable plan can be found in the plan library.

- The representation problem has to be discussed on two levels: on the representation level of a single plan or plan schemata entry and on the representation level of the whole plan library.

- Two views on plan determination have been analyzed: the analogy-driven plan determination and the efficiency-driven plan determination. Both views are different in the evaluation function they apply to determine a reuse candidate from the plan library. In the analogy-driven approach, a similarity metric is used to compare the various planning situations, while in the efficiency-driven approach, an estimation of the modification effort is applied. To decide which approach is more appropriate the underlying plan generation formalism and the properties of the application domain have to be considered. A combination of both approaches might be useful.

- The modification of plans has been studied by various authors (cf. [Alt86], [Ham90], [Kam89a], [Wil83]) and leads to a large collection of heuristic failure recognition procedures and corresponding refitting strategies. A logical formalization of this process has not been achieved.

The analysis of the plan reuse process has shown that a logic-based plan reuse system has to cover:

- the representation of goals, plans and information from the plan generation process, e.g., applied transformation rules or resulting verification conditions,

- the matching and instantiation of plan schemata in the determination of an appropriate reuse candidate,

- the proof, whether a reuse candidate is a solution to the current planning problem or not,

- the modification of reuse candidates based on results provided by the planner, e.g., the linearization of subgoals and the integration of newly generated subplans into the reused plan,

- the verification of the modified plan by which the plan is proved to be a correct solution,

- the generation of a new plan library entry as a process of abstraction and generalization, and

- the update of the plan library as a process of explanation-based learning or inductive learning.

Other elements of the reuse process will require a heuristic, knowledge based component covering:

- the representation of the plan library reflecting types of planning problems in the application domain,

- the heuristic search in the plan library based on an evaluation function,

- the determination of a reuse candidate as an activation of explicitly stored (compiled) problem solving knowledge,

- the control of the modification process by heuristic refitting strategies based on plan failure classifications,
- the generation of a new plan library entry based on heuristics about the frequency and importance of planning problems occuring in the application domain, and
- the update of the plan library as a process of heuristic classification.

5.2 Architecture of a Plan Reuse System

In the previous chapters we discussed the various subproblems concerning plan reuse. As a result, we now propose a 4-phase model that reflects these subproblems and supports a structured treatment of plan reuse. In figure 2 the architecture of a plan generation mechanism with an integrated reuse facility based on the 4-phase model is shown.

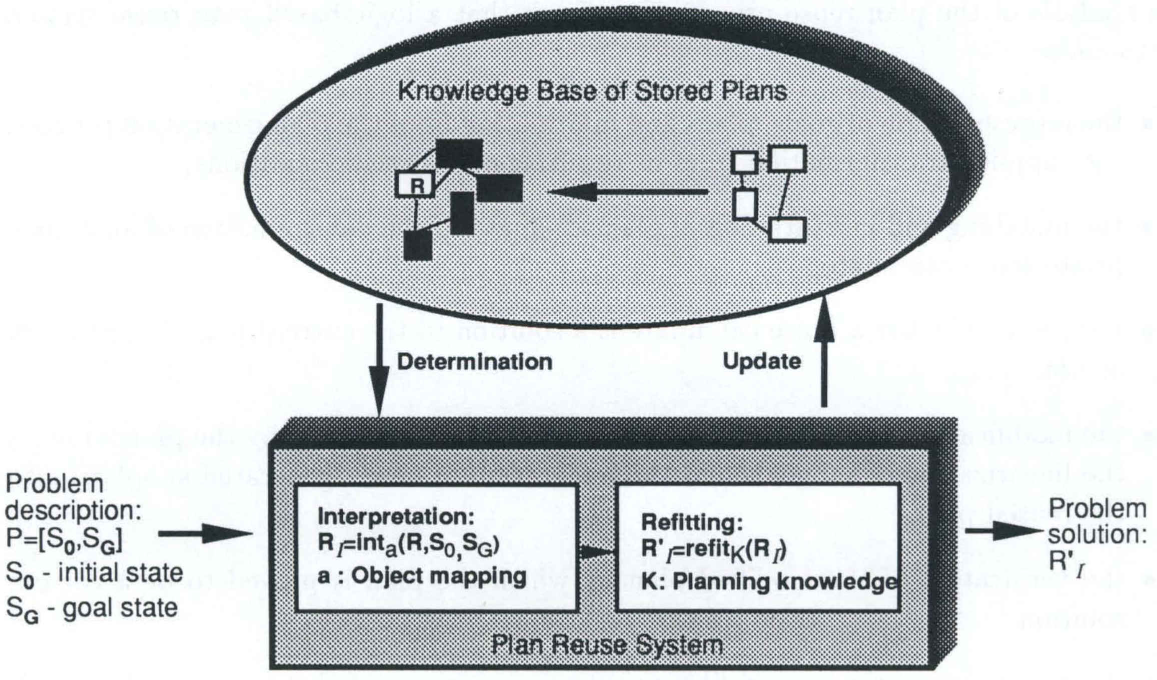

Figure 2: A 4-Phase Model of Plan Reuse

The underlying planning system is based on a program synthesis procedure that was originally developed for the automation of existential proofs [Biu91]. A plan is generated by proving a specification formula in first order predicate logic:

$$\psi = \forall x^* : I(x^*, s_0) \rightarrow \exists z : G(x^*, s_0!z)$$

$I(x^*, s_0)$ represents the formulas that hold in the initial state and $G(x^*, s_0!z)$ represents the formulas that hold in the goal state s_G, which is obtained by executing the plan z starting in s_0. The sentence ψ is the *current problem description* for the planner.

After skolemization of the existence formula, a number of transformation rules are applied to construct a term representing the specified plan. As a result of the transformation process two sets of formulas arise: a set of conditional equations (representing the plan) and a set of formulas that can be viewed as assertions of verification conditions for the plan. They describe additional properties that have to hold in the domain theory to make the plan successfully applicable.

We intend to develop a deductive plan reuse mechanism that mainly relies on the verification conditions initially provided by the planner: if a verification condition can be proved to hold in a particular situation, the plan that was obtained as a solution of the corresponding specification formula is applicable in the particular situation and it reaches the intended goal.

The reuse mechanism will proceed in the following 4 phases:

1. **Determination:**
   The reuse system is provided with a problem specification $[S_0, S_G]$ represented by the above described specification formula. To solve the planning problem, a stored plan entry from the plan library is determined. We presuppose that the plan library does not contain (user-) predefined plan entries, but is built up using information provided by the deductive plan generation component. This information consists of:

   (a) V_0: the set of verification conditions for the plan,

   (b) G_0: the set of formulas representing the goal state,

   (c) S_0: the set of formulas representing the initial state,

   (d) P_0: the set of formulas representing the plan,

   (e) O_0: the set of objects involved in the plan.

   The determination of a plan entry R can be based on syntactic comparisons of (I, S_0), (G, G_0), and (x^*, O_0).

2. **Interpretation:**
   Closely related to the determination phase is the interpretation of R in the new planning situation. The main problem here is to find the correct mapping a of objects in x^* and O_0 and the correct instantiation of P_0 (if formula schemata are considered). A possible, but not sufficient criterion can be that P_0 is instantiated such that, as much as possible, formulas from G are true in the instantiated plan schema.

3. **Refitting:**
   The interpretation process provides a (partially) instantiated plan schema together with (partially) instantiated verification conditions.

   An analysis of G and the interpreted formulas in G_0 allows G to be split into 3 subgoal sets, if G is not an atomic formula:

   - $G_P = I \cap G$ the set of phantom goals that already hold in the initial state,
   - $G_R = Int_a(G_0)$ the set of subgoals solved by the reused plan,
   - $G_N = G \setminus (G_R \cup G_P)$ the set of subgoals for which the planner has to be activated.

The applicability of the reused plan can be verified based on the verification conditions from V_0. As a result of this phase, a solution to the planning problem based on the reuse of a stored plan will be obtained.

4. **Update:**
   If a problem solution $z = R'_I$ is obtained, a new plan library entry can be built up from it:

   - G_0, S_0, P_0 can be determined based on G, I, and z, respectively,
   - V_0 can be determined based on the verification conditions of the subplans for G_P, G_R, G_N,
   - O_0 is obtained on the basis of x^*.

   To decide whether a plan is "worth" storing in the plan library, the same similarity metrics or efficiency measures used in the determination phase will be applied.

Appendix

Examples of Plan Reuse in Implemented Systems

A KIP - Knowledge Intensive Planning

Luria [Lur88] describes the development of the Knowledge Intensive Planner KIP, which is the planning component for UC (Unix Consultant), an intelligent Unix help system that allows a natural language interaction with the user. KIP is based on Wilensky's idea of knowledge intensive planning [Wil83]. It uses the identification of different goal interactions to classify potential plan failures and implements some aspects of the meta-planning theory as described in section 4.1.1.

KIP's process of generating a plan is called plan synthesis and consists of an iterative process composed of three parts:

1. *Goal establishment* describes the phase in which the goals of the user with whom UC interacts are determined. The goals for which KIP searches are derived from these user goals.
2. *Plan determination* aims at the specification and retrieval of a plan that meets the intended goals of the user.
3. *Plan failure detection* tests whether the plan will work in the current problem situation without causing unacceptable consequences, and modifies the plan if necessary.

After a successful completion of all three parts the plan is used by the help system to support the user.

The major problem that is addressed by KIP is how to focus on relevant knowledge, i.e., in particular how to exploit the represented knowledge to identify potential plan failures. As a new idea, so-called *concerns* are introduced to identify which aspects of a plan are most likely to fail.

A.1 Representation of Plans

KIP's knowledge representation is based on the following assumptions:

- Plans are ordered according to the goals they fulfil and it is possible to determine the best plan for a particular goal.
- The plan determination algorithm is *knowledge efficient.* That means the system is able to exploit the knowledge base in such a way that it considers only those pieces of knowledge that are relevant in the current planning situation. This assumption helps to overcome problems of combinatorial explosion during search processes.

- Knowledge can be interpreted in a context-sensitive way, i.e., defaults exist that can be adapted on various situations.

To implement these assumptions in KIP the KODIAK knowledge representation language [Lur88] has been developed that supports the representation of taxonomic hierarchies in a KL-ONE like manner.

Goals and plans are represented as concepts and are related by the PLANFOR-relation. The determination of the user's goals is performed by the PAGAN goal analyzer [Lur88] with which KIP is interacting. PAGAN transforms the natural language input so that it can be directly used for the search in the knowledge base. The following example is taken from [Lur88]:

The user inputs "How do I rename the file named lewis to be called bernstein?". PAGAN analyzes this utterance and transforms it into the goal for which KIP has to search a plan:

```
(rename-file-effect-1
          (New-Name-37 bernstein-1)
          (Previous-Name-63 lewis-1)
          (Destination-File-Of-Rename-File-Effect-48 file-2))
```

The goal is matched to a goal concept in the hierarchy. If it is a complex goal, KIP has to decompose it into elementary goals. For example, "Print-and-Delete-File" is a complex goal that is decomposed into two goals "Print-File" and "Delete-File." For each elementary goal an *importance level* is stored representing the importance of the goal in relation to other goals that can simultaneously occur. This leads to a ranking of subgoals that determines the order in which plans are searched for meeting these subgoals.

The goal concept *Rename-File-Effect* is an elementary goal and is therefore directly linked with a list of plan schemata from which one plan schema is chosen and instantiated to meet the current goal. In this example, this leads to the plan schema:

```
(mv-command-71
          (First-File-Arg-File-Name-100
                    (file-name-state-92
                              (Value-of-File-Name-98 lewis-1)
                    (Object-of-File-Name-96 file-1)))
          (Second-File-Arg-File-Name-83
                    (file-name-state-75
                              (Value-of-File-Name-81 bernstein-1)
                    (Object-of-File-Name-79 file-2))))
```

To recognize plan failures, knowledge describing these failures is represented by *concerns* that have been developed especially to perform plan failure detection. They refer to those aspects of a plan that are sources of possible plan failures and that are associated explicitly with every plan schema stored in the knowledge base.

Concerns introduce probabilistic reasoning into the planning mechanism. They represent a heuristic approach to reduce the complexity of the search space when the effects of an action are computed since they restrict the computation on conditions and effects which are considered as important in the current situation. It is this degree of importance that is represented in the concerns.

Stored concerns are a means for the plan library designer to express which aspects of a stored plan schema are most likely to fail independently of the current planning situation. Applying stored concerns in the planning situation at hand is done by *dynamic concerns*. They are instances of stored concerns and represent the inheritance of concerns among plan schemata and their corresponding plan instances. Dynamic concerns are introduced by the system as soon as it notices a potential condition or goal conflict failure.

The plan library designer has to estimate two factors: the probability that a particular precondition will not be satisfied and the probability that an unsatisfied precondition causes the plan failure. This is represented by assigning a number between 1 and 10 to every precondition.

A.2 Plan Determination

Plans are generated by retrieving and instantiating fixed, unchangeable plan schemata from the plan library. This process of plan determination consists of two phases:

- *Plan selection*: KIP exploits the hierarchy of goals to find a stored plan schema associated with a goal. Every goal in the plan library has a PLANFOR-relation that connects it to plan schemata that solve this goal. These plan schemata are selected and the first is instantiated. If no plan schema can be found for the goal considered, a similar goal is searched for using the *Goal Similarity Matching Algorithm* (GSMA) [Lur88]. To find a goal in the hierarchy that is most similar to the current goal, KIP considers the goal concepts that are represented at the same level in the taxonomic hierarchy and selects the one which belongs to the smallest subgraph containing both goals.

- *Plan specification*: After a plan schema is determined it has to be instantiated due to the current planning situation. This instantiation process is guided by information stored in so-called EQUATE-associations between the unspecified values in the plan schema and the specific values in the goal.

After instantiating the plan, KIP decides if the plan will work in the current situation using the plan failure detection mechanism. If a plan only partially solves a goal additional subplans are added. If the plan leads to undesired effects subplans that remove these effects are retrieved from the plan library.

A.3 Plan Failure Detection

If KIP has to find a plan for a complex goal, it decomposes this goal and determines plans for the particular subgoals based on the PLANFOR-relations. It has to be investigated whether these subplans interact without causing failures. Two possible kinds of failures are distinguished:

- *Condition failures* occur when a necessary precondition for an action is unsatisfied in the world state. To solve a condition failure the satisfaction of the condition becomes a new goal for KIP. If the planner cannot find a plan to meet the additional goals, a new plan must be determined.

- *Goal conflict failures* occur when a goal is incompatible with goals the selected plan meets. Three different strategies for solving goal conflicts are used in KIP that modify the plan to avoid the goal conflict, determine a new plan, and abandon one of the user's goal, respectively.

KIP requires that all the preconditions necessary for plans to be successfully executed are explicitly represented because it evaluates these preconditions in the current environment to discover potential plan failures. The detection of goal conflicts entails the comparison of every effect of a plan with every goal owned by the user. To avoid combinatorial explosion, concerns are applied that enable the planner to concentrate on conditions and effects which are most likely to fail.

A.4 Evaluation of KIP

We ground the evaluation on the model of plan reuse as developed in section 5.2:

- Representation of plan entry and plan library:
  Plan schemata are connected to the goals they achieve and are organized in an inheritance network. The KL-ONE like representation is augmented by probabilistic elements to increase the efficiency of the failure detection and refitting process. A theoretical basis for the probabilistic reasoning is not provided.

- Underlying plan generation mechanism:
  Plan generation is reduced to the retrieval of plan schemata from the knowledge base and the instantiation of variables depending on the current planning situation. It is always treated as the activation of stored plan schemata. There is no mechanism to generate a new plan from available operators.

- Determination of plans from the plan library:
  To determine a plan KIP uses a similarity based matching algorithm. This requires the exhaustive description of the current goal and situation with attributes known by the system. Exhaustive goal descriptions are necessary to find the best plan to achieve the goal. Exhaustive situation descriptions are necessary to investigate the most important concerns to prevent the plan from failing.

- Interpretation of a reused plan in the new planning situation:
  The correct instantiation of plans is guided by knowledge about object classes, equality and inheritance relations.

- Refitting of the reused plan:
  Luria suggests a classification of potential plan failures based on the approach by Wilensky (cf. [Wil83]) and demonstrates general strategies for refitting these failures.

- Update of the plan library:
  KIP is explicitly provided with a fixed plan library from which it retrieves the reused plans. An update mechanism is beyond the scope of the approach.

B CHEF - Case-based Planning

We review the work of Hammond [Ham86b], [Ham86a] and [Ham90] describing CHEF, a system working in the domain of cooking. Plans are recipes that meet goals representing user demands on the dishes to be cooked. Reusing plans presupposes the understanding and explanation of why plans failed or succeeded in a particular situation. Therefore, case-based reasoning systems need a strong model of causality of the domain in which they operate. While KIP is supplied with a large library of plan schemata for nearly every goal and only instantiates theses schemata, CHEF uses plan failures to modify even its model of the world that led to the creation of a faulty plan.

B.1 Representation of Plans

The memory of CHEF consists of three main parts:

- a description of past successes, i.e., plans that have been succesfully applied,
- a taxonomy of past failure descriptions that warn the planner to avoid these failures if a similar situation occurs again,
- a taxonomy of repair strategies ordered by plan failures that is used to deal with these failures.

As in KIP, plans are first of all indexed by the goals they meet. But a second index orders them by the failures they avoid. Failures are indexed by the features of the current planning situation that predict them. The aim of this representation principle is to anticipate and avoid future plan failures.

On the implementational level plans are organized in discrimination networks. Elements of object-oriented representation like demons are added to the net nodes that represent goals. Demons are used to implement repair strategies and to provide an implicit representation of repair knowledge. Each goal node is connected to plans that meet this goal and has links to typical failures that might occur when the goal is achieved with one of the plans.

B.2 Plan Determination

CHEF analyzes the features of the planning situation and deduces which potential failures these features predict. Then it uses this information to find a plan that solves the intended goals and also avoids the predicted failures. A predicted failure is transformed into a new goal. The achievement of this new goal avoids the failure. This new goal is added to the goal list containing all the goals that have to be met in the current planning situation.

CHEF now looks for a plan that achieves all goals originating from potential failures and as many as possible goals stated by the user. It is worth noting that goals coming from the need to avoid failures are more important for the determination of a plan than the original goals for which a plan is searched for.

Plan retrieval uses a *best match* strategy. Best match is defined in CHEF as finding a plan that satisfies or partially satisfies as many goals as possible.

The plan retrieval component also uses a similarity metric to recognize partial matches between goals and a value hierarchy to judge the importance of the goals it plans for. The most important goals are the ones describing the kind of dish that has to be cooked, followed by the goals arising from failures that have to be avoided, again followed by the goals describing the incredients that have to be included in the dish. If a plan is found, but not all goals stated by the user can be achieved performing the plan, it has to be modified.

B.3 Plan Failure Detection

Two situations where plan modification is necessary are considered. The first occurs when the plan does not meet all the user's goals. The second occurs when the plan fails during its simulation.

If a plan fails, a repair strategy is used to create a causal explanation for that failure. This causal explanation directs the repair process of the plan and helps to recognize the features of the planning situation that caused the failure.

Generating an explanation for a failure means backchaining from the failure to the initial steps or states in the world based on rules representing possible consequences of actions under various circumstances.

Explanations are connected to so-called *Thematic Organization Packets* (TOPs) that are indexed by the description of a planning problem type. TOPs were developed by Schank [Sch82]. Each packet contains a set of strategies to deal with a particular type of problem. These strategies are general repair rules based on information about the interaction of plans and goals and the interaction among planning steps.

Hammond develops a very detailed taxonomy of plan failures that are characterized by operators and states in the domain. Five categories of different planning problems are described on the top level of the taxonomy. They represent general problems concerning side effects of planning steps interacting with other planning steps, unwanted features of objects, or blocked preconditions. These general failure categories are further analyzed and refined leading to a detailed conflict description on the bottom level where these conflicts are linked with subsets of instances of 17 general repair strategies that can be applied to resolve the conflicts. The TOPs are organized such that different conflicts can be resolved by the same general repair strategy but require different instantiations.

The repair strategies in general lead to a reordering of steps, an alteration of the objects involved, and a change of actions, respectively, to resolve the conflicts.

TOPs are also used to modify the domain model by marking the features that caused the failures. The explanation is combined with the *most generalized description* of the object that was involved in the failure situation. The explanation is used to derive tests for potential plan failures connected to the object if it occurs again in a goal.

A detailed description of failures and repair strategies can be found in [Ham90].

Once a plan is repaired, it can be described as a plan that avoids the problem at hand and achieves the original goal. Modification rules have to be applied if a plan does not already meet all original goals. The library of modification rules is indexed by the changes that have to be made and by the type of plans in which the changes have to be made. Modification rules are descriptions of steps that have to be added and deleted from plans to satisfy the new goals. They are represented in an abstraction hierarchy. If no modification rule for a goal can be found, a rule to achieve the more general goal is searched for.

Object critics describe failures related to the ingredients of the dishes (the objects of the domain). Object critics are used to guide the modification process when new objects are introduced instead of the ones for which the plan was created. When one ingredient is replaced by another, the object critics remove all planning steps corresponding to the old object critic and add steps from the new object critic instead.

Each plan is carried out by a simulator that runs the plan. It uses a set of rules describing the effects of each action in the domain and thus evaluates whether an action achieves the intended goal. Every plan that is executed successfully is added to the knowledge base. Plans are stored in their original form. There is no generalization or abstraction.

B.4 Evaluation of CHEF

- Representation of plan entry and plan library:
  The underlying knowledge representation formalism combines discrimination networks and frames by which the nodes of the network are represented.

- Underlying plan generation mechanism:
  CHEF is an application of case-based reasoning to planning. New plans are generated by retrieving and modifying stored plans. A generation of plans based on operators is not intended in the field of case-based planning.

- Determination of plans from the plan library:
  The explicit representation of failures a plan might cause and the use of these failures as an indexing mechanism when a plan is searched in the knowledge base is an interesting idea. CHEF analyzes the user's goal and tries to anticipate potential failures this goal might include. Then it searches for a plan that avoids these failures and meets the original goal. It is worth noting that the metagoal of avoiding failures is more important for the selection of a plan than the original user's goal. The anticipation of failures requires their explicit representation that is described by Hammond as the "deep model of causality" on which CHEF is based.

- Interpretation of a reused plan in the new planning situation:
  The reused plan is simulated in the world model, which can be considered as a variant of interpretation. During this simulation the information is collected that guides the refitting of the plan.

- Refitting of the reused plan:
  CHEF follows the *Generate, Test and Debug* paradigm in which plan failures are discovered during plan execution when interactions among planning steps become obvious. The complexity that arises when projecting the effects of an action is avoided by CHEF using a heuristically driven process of failure anticipation. The refitting process is an example of plan debugging, rather than plan modification as discussed in section 4. The very detailed classification of plan failures and corresponding repair strategies provides a good basis for further research.

- Update of the plan library:
  If CHEF has solved a planning problem, it stores the modified plan resulting from the reuse candidate in the case base. Plans are stored in their original form. There is no generalization or abstraction. The system is also able to reorganize the plan library so that it stores information about plan failures together with the plans. This leads to a changed indexing of the plans and can be viewed as a change in the world model CHEF exploits.

C PRIAR - Integration of Plan Generation and Plan Reuse

Kamphampati [Kam89a] tries to develop a formal treatment of plan reuse using a hierarchical planning formalism. The main problem that is addressed with PRIAR is how plans have to be modified for reuse in a new planning situation.

C.1 Representation of Plans

The distinguishing feature of the representation formalism is the augmentation of the plan representation with additional information that represents a hierarchical explanation of correctness for a generated plan. This information is the so-called *validation structure* and is annotated with every plan the planner generates. Since the information that constitutes the validation structure is automatically extracted from the planning knowledge, the approach is flexible and domain independent.

Basic representational concepts that stem from the underlying hierarchical planner are:

- The *hierarchical task network* represents the development process of each plan. A hierarchical planner is one that starts with an abstract specification of a plan and then refines it until the plan is reduced to a sequence of applicable actions.

- *Task reduction schemata* represent the rules the planner uses to reduce the abstract plan to a sequence of applicable actions.

- *Task-kernels* represent the set of conditions that have to be preserved by any schema instance that is chosen by the planner to reduce the refit-task. Three different types of conditions are distinguished:

  - *effect conditions* describe the effects of the refit-task used in other parts of the plan,

  - *persistence conditions* describe conditions that have to hold over the whole plan,
  - *external preconditions* describe conditions that were introduced by other planning steps and that have to be met by the schema instance.

The following representational concepts are added to augment the hierarchical planning formalism with the ability to reuse plans:

- The *validation structure* represents why a plan is correct, storing causal dependencies between the planning steps and the underlying planning decisions. It is a 4-tuple $\langle E, n_s, C, n_d \rangle$, stating that the effect E of a source action n_s supports the condition C that destination action n_d requires. The validation structure is stored in the form of plan kernels.

- *Plan kernels* contain those features of the initial situation and the goal situation of a plan that provide validations for parts of the plan. Three types of validations are distinguished ordered by their importance:

  - features that validate the goals of the plan,
  - features that validate filter conditions or top level phantom goals (cf. section C.2),
  - features that validate preconditions.

  If validations are violated that are necessary to achieve the goal, a number of extra subgoals have to be added to reestablish the validations. If validations for filter conditions or preconditions are violated, it will be easier to establish the failing validations.

- The *justification structure* represents the underlying preference rules for planning choices if the same state can be achieved by different actions.

- *Annotations* are connected to the hierarchical task network and describe the internal causal and decision dependency structure of the plan. Two types of annotations are used:

  - *Node annotations* are linked with the nodes in the hierarchical task network and represent dependencies between the sub-reductions rooted at the node and the remaining plan. They contain the information that is necessary to guide the modification process and predict the utility of reusing the plan in dependence of features in the input and output situation of that plan.
  - *Annotation states* connect two successive plan steps and represent the conditions that have to hold after the first plan step is performed to ensure that the following plan step is applicable. They are used to detect applicability failures of the retrieved plan in the current planning situation.

- *Refit tasks* represent domain-independent modification strategies.

C.2 Plan Determination

The retrieval mechanism searches for a plan that requires the least modification effort when applied in the current planning situation. To measure the modification effort, the validation structure of the plan (as described in section C.1) is considered under the current planning conditions. The plan with the smallest number of validations that fail in this situation is preferred for reuse. The plan determination process proceeds in the following way:

The system is provided with a new problem description $P^n = [I^n, G^n]$ (where I^n describes the inital state, and G^n describes the goal state) and a set of reuse candidates $\{\langle R^0, \alpha\rangle\}$ where R^0 is the candidate plan and α is a mapping of the corresponding objects in P^n and R^0.

The plan kernel of R^0 written as $PK(R^0)$ that represents the validation structure of the candidate plan is interpreted in the new planning situation. It contains three sets of features:

$$PK(R^0) = \langle \textit{g-features, f-features, pc-features} \rangle$$

The **g-features** (Goal Features) correspond to validations of R^0 that support its goal, i.e., if they hold in the current planning situation the plan will achieve its goal. The system only considers candidate plans that solve the current goal, since the candidate set of potential applicable plans is determined by an unification of goal states. To verify the goal validations, the retrieval mechanism tests whether the set of g-features is equal to the set of validations for the goal node $A^P(n_g)$:

$$\textit{g-features}(PK(R^0)) = A^P(n_g)$$

The candidate plans that fulfill this condition are considered in a second test. The other plans are dropped because they would require the addition of an extra goal to solve the new planning problem. Now the set of **f-features** (Filter Features) of the remaining candidate plans is considered. These features correspond to the validations for the input specification of R^0 and describe those preconditions the candidate plan requires to be applicable and for which no actions exists in the planning domain. They are a-priori conditions that must hold in the planning situation. Filter conditions usually describe features of objects that are relevant for the planner and which are invariant domain constraints the planner has to meet.

After this test, the set of candidate plans is further reduced. If it still contains more than one plan, a third test has to be performed to produce a final ranking of the candidates. This last test considers the **pc-features** (Precondition Features) that correspond to the validations that support the preconditions of the actions in the candidate plan. The candidate plans are ranked according to the number of preconditions that might fail in the new interpretation, with the plan having the least number of failing preconditions on the top of the ranking.

The verification tests for the three types of validations are implemented using a syntactic match of features.

This idea of validated retrieval reflects the following heuristics:

- A plan can be reused efficiently by minimizing the number of necessary modifications.
- The modifications can be recognized considering three different types of validations reflecting three types of plan failures:
    1. The plan only partially meets the new goal.
    2. The plan uses the wrong types of objects that are determined by the mapping α in the new planning situation.
    3. The plan requires preconditions that do not hold in the new planning situation.
- The different types of plan failures lead to differing amounts of necessary effort to overcome these failures:
    1. The first kind of plan failures can only be corrected by adding a new goal to the list of goals. This requires the generation and integration of an additional subplan.
    2. The second kind of plan failures can be solved if some of the objects are exchanged by other objects that meet the filter conditions. This requires the automated construction of a new mapping α.
    3. The third group of plan failures requires the least modification effort, since it can be assumed that failing preconditions often can be easily established performing some of the elementary actions the planner uses.

In every step the retrieval mechanism drops the plans that require the highest modification effort. If it terminates with a solution, this represents the plan from the plan library that fits best to the new planning situation. As a side-effect, the retrieval mechanism additionally collects the information that will guide the modification process.

C.3 Plan Failure Detection

In a first step, the candidate plan is interpreted using the mapping α from the plan determination phase. Then the differences between the specifications of the old and the new planning situations are marked to focus the annotation verification procedure on the inconsistencies in the validation structure of the candidate plan.

Old and new planning situations are specified by their initial and goal states. The differences between the initial states and the goal states of the two situations are represented in two sets of facts I_i and G_i. The differences in I_i and G_i may lead to inconsistencies in the validation structure of the interpreted candidate plan R_i.

If I_i and G_i are empty the plan does not have to be modified. Otherwise, I_i may contain facts marked as *out* representing those facts which no longer hold true in the current situation, and facts marked as *new* that now come into existence, but did not hold in the old situation. G_i may contain *extra goals* representing the goal state facts that were introduced by the new planning goal and *unnecessary goals* which remain from the old goal description but are not required by the new goal.

The inconsistencies in the validation structure represent different classes of applicability failures. The annotation verification procedure determines these classes of failures based on the type of inconsistency and proposes a refitting strategy for every class. The following classes of inconsistencies are distinguished:

1. *Unnecessary validations*: All validations that support conditions which are no longer required can be removed, including all parts of the plan of which the sole purpose is supplying these validations.

2. *Missing validations*: All facts G that represent extra goals in G_i are transformed into corresponding refit tasks of the form $Achieve(G)$, for which the planner has to be activated. If a new validation is added, the planner has to check the validation structure for harmful interactions. The extra goal is added to the plan at a place where it causes the fewest interactions.

3. *Failing validations*: The "out"-facts in I_i are further classified and treated separately:

   - *Failing precondition validations*: Failing preconditions are achieved by introducing an extra goal that achieves the supporting effect for this precondition. The extra goal is inserted before the destination node.
   - *Failing phantom validations*: If a validation for a phantom goal is failing, the marking of the goal is undone, telling the planner that this goal has to be achieved. The failing validation for this goal is removed.
   - *Failing filter condition validations*: Failing filter conditions cannot be achieved by the planner by establishing additional goals. Therefore the planning conditions that introduced these failing filter conditions have to be undone, i.e., the whole subreduction of a goal has to be exchanged for an alternate schema instance. If more than one schema instance is available, the one which least affects the already established validations is chosen.

4. *P-phantom validations*: All validations whose source node is not the initial node of the hierarchical task network are checked whether they could have been exchanged for validations that were added to the plan during the annotation verification. These new validations arise from the extra goals the planner has to achieve and are necessary for a consistent validation structure. If they contain validation information that was already established by old validations these old validations together with their corresponding planning steps can be removed.

In the refitting phase the planner takes the verified plan net consisting of the applicable parts of the old plan and the refit tasks that were added during the verification phase. The refit tasks are reduced applying a conservative control strategy, i.e., the planner selects those schema instances to reduce the refit tasks that minimally disturb the validation structure of the remaining plan.

C.4 Evaluation of PRIAR

- Representation of plan entry and plan library:
  PRIAR uses a knowledge representation that is especially developed for the purpose of plan reuse. An analysis of the reuse process leads to a graph-based representation reflecting the internal dependencies among the planning steps in a plan. The representation and organization of the whole plan library is beyond the scope of the approach.

- Underlying plan generation mechanism:
  PRIAR is based on a hierarchical, nonlinear planning mechanism. It is the first approach towards an integration of plan generation and plan reuse and is closely related to our model of plan reuse.

- Determination of plans from the plan library:
  The approach concentrates on the investigation of the modification phase in the reuse process. Therefore the reuse candidates are explicitly provided to the reuse mechanism. The approach lacks a search procedure that is able to select the initial candidate set of potential applicable plans from a large plan library.

- Interpretation of a reused plan in the new planning situation:
  The basis for the interpretation is a mapping α of corresponding objects that has to be provided. The system is not able to discover exisiting relations or similarities between plans by itself.

- Refitting of the reused plan:
  This subphase is extensively studied in PRIAR. The system is able to detect necessary modifications based on occuring inconsistencies in the validation structure. An analysis of these inconsistencies leads to a classification of potential plan failures. Refit tasks that represent goals for the planner are used to remove the inconsistencies. The reuse component guides the modification process, but the modification of the plan itself is performed by the planner in solving the refit tasks. The analysis of this interaction in plan generation and plan reuse is the main contribution of PRIAR to the problem of plan reuse from our perspective.

- Update of the plan library:
  The approach does not address the organization of plans in the plan library and other problems related to the update of the plan library as, e.g., the generalization of stored plans. In principle, an update mechanism seems to be possible, since all the information that is necessary to store a plan entry can be extracted from the plan generation process but is not provided in the implementation.

References

[AF88] J.S. Anderson and A.M. Farley. Plan abstraction based on operator generalization. In *Proceedings of the 7th National Conference on Artificial Intelli-*

gence, Saint Paul, Minnesota, USA, pages 100–104. Morgan Kaufman, San Mateo, 1988.

[Alt86] R. Alterman. An adaptive planner. In *Proceedings of the 5th National Conference on Artificial Intelligence, Philadelphia, USA*, pages 65–69. Morgan Kaufman, Los Altos, 1986.

[Bak89] A.B. Baker. A simple solution to the Yale shooting problem. In *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning, Toronto, Ontario, Canada*, pages 11–20. Morgan Kaufman, Los Altos, 1989.

[BCP90] B. Brock, S. Cooper, and W. Pierce. Analogical reasoning and proof discovery. In *Proceedings of the 9th Conference on Automated Deduction, Kaiserslautern, Germany*, pages 454–468. Lecture Notes in Computer Science 310, Springer, Berlin, 1990.

[BH90] G. Brewka and J. Hertzberg. How to do things with worlds: On formalizing actions and plans. Preliminary Draft, 1990.

[Biu90] S. Biundo. Plan generation using a method of deductive program synthesis. Research Report RR-90-09, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, 1990.

[Biu91] S. Biundo. *Automatische Synthese rekursiver Algorithmen als Beweisverfahren.* Informatik Fachberichte. Springer, Berlin, 1991. forthcoming.

[BL88] S. Bradtke and W.G. Lehnert. Some experiments with case-based search. In *Proceedings of the 7th National Conference on Artificial Intelligence, Saint Paul, Minnesota, USA*, pages 133–138. Morgan Kaufman, San Mateo, 1988.

[BM79] R.S. Boyer and J S. Moore. *A Computational Logic.* ACM Monograph Series. Academic Press, London, 1979.

[Bra90] R.J. Brachman. The future of knowledge representation. In *Proceedings of the 8th National Conference on Artificial Intelligence, Boston, USA*, pages 1082–1092. The MIT Press Menlo Park, Cambridge, London, 1990.

[Bun90] A. Bundy. The use of explicit plans to guide inductive proofs. In *Proceedings of the 9th Conference on Automated Deduction, Kaiserslautern, Germany*, pages 111–120. Lecture Notes in Computer Science 310, Springer, Berlin, 1990.

[Bür86] H.J. Bürckert. Lazy theory unification in prolog: An extension of the warren abstract machine. In *Proceedings of the 10th German Workshop on Artificial Intelligence and the 2nd Österreichische Artificial Intelligence Tagung, Ottenstein, Austria*, pages 277–288. Informatik-Fachberichte 124, Springer, Berlin, 1986.

[C+86] R.L. Constable et al. *Implementing Mathematics with the NuPRL Proof Development System.* Prentice Hall, 1986.

[Chi89] S.A. Chien. Using and refining simplifications: Explanation-based learning of plans in intractable domains. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence, Detroit, Michigan, USA*, pages 590–595. Morgan Kaufman, San Mateo, 1989.

[CI89] J. Cheng and K.B. Irani. Ordering problem subgoals. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence, Detroit, Michigan, USA*, pages 931–937. Morgan Kaufman, San Mateo, 1989.

[DC89] M. Drummond and K. Currie. Goal ordering in partially ordered plans. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence, Detroit, Michigan, USA*, pages 960–965. Morgan Kaufman, San Mateo, 1989.

[Dem67] A.P. Dempster. Upper and lower probabilities induced by a multivalued mapping. *Annals of Mathematical Statistics*, 38:325–339, 1967.

[DHR89] F. Daube and B. Hayes-Roth. A case-based mechanical redesign system. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence, Detroit, Michigan, USA*, pages 1402–1407. Morgan Kaufman, San Mateo, 1989.

[DR87] T.R. Davis and S.J. Russell. A logical approach to reasoning by analogy. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence, Milan, Italy*, pages 264–270. Morgan Kaufman, Los Altos, 1987.

[Eth88] D.W. Etherington. *Reasoning with Incomplete Information.* Morgan Kaufman, Los Altos, 1988.

[FF86] B. Falkenhainer and K.D. Forbus. The structure-mapping engine. In *Proceedings of the 5th National Conference on Artificial Intelligence, Philadelphia, USA*, pages 272–277. Morgan Kaufman, Los Altos, 1986.

[FHN71] R.E. Fikes, P. Hart, and N.J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1971.

[Fin87] J.J. Finger. *Exploiting Constraints in Design Synthesis.* PhD thesis, Stanford University, Stanford, CA, 1987.

[FN71] R.E. Fikes and N.J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189 –208, 1971.

[Gen88] D. Gentner. Structure-mapping: A theoretical framework for analogy. In *Readings in Cognitive Science: A Perspective from Cognitive Science and Artificial Intelligence*, pages 303–310. Morgan Kaufman, San Mateo, 1988.

[GM88] P. Gärdenfors and D. Makinson. Revisons of knowledge systems using epistemic entrenchment. In *Proceedings of the 2nd Conference on Theoretical Aspects of Reasoning about Knowledge, Pacific Grove, California, USA*, pages 83–95. Morgan Kaufman, Los Altos, 1988.

[Gre88] R. Greiner. Learning by understanding analogies. *Artificial Intelligence*, 35:81–125, 1988.

[GS88a] M.L. Ginsberg and D.E. Smith. Reasoning about action I: A possible worlds approach. *Artificial Intelligence*, 35:165 - 195, 1988.

[GS88b] M.L. Ginsberg and D.E. Smith. Reasoning about action II: The qualification problem. *Artificial Intelligence*, 35:311 - 342, 1988.

[HA96] M. Haraguchi and S. Arikawa. Reasoning by analogy as a partial identity between models. In *Proceedings of the International Workshop on Analogical and Inductive Inference, Wendisch-Rietz, GDR*, pages 61–87. Lecture Notes in Computer Science 265, Springer, Berlin, 1896.

[Hal89] R.P. Hall. Computational approaches to analogical reasoning: A comparative analysis. *Artificial Intelligence*, 39:39–120, 1989.

[Ham86a] K.J. Hammond. CHEF: A model of case-based planning. In *Proceedings of the 5th National Conference on Artificial Intelligence, Philadelphia, USA*, pages 267–271. Morgan Kaufman, Los Altos, 1986.

[Ham86b] K.J. Hammond. Learning to anticipate and avoid planning problems through the explanation of failures. In *Proceedings of the 5th National Conference on Artificial Intelligence, Philadelphia, USA*, pages 556–560. Morgan Kaufman, Los Altos, 1986.

[Ham88] K.J. Hammond. Learning from opportunities: Storing and re-using execution-time optimizations. In *Proceedings of the 7th National Conference on Artificial Intelligence, Saint Paul, Minnesota, USA*, pages 536–540. Morgan Kaufman, San Mateo, 1988.

[Ham89] K.J. Hammond. CHEF. In C.K. Riesbeck and R.C. Schank, editors, *Inside Case-based Reasoning*, chapter 6. Lawrence Erlbaum, Hillsdale, New Jersey, 1989.

[Ham90] K.J. Hammond. Explaining and repairing plans that fail. *Artificial Intelligence*, 45:173–228, 1990.

[Hay89] C.C. Hayes. A model of planning for plan efficiency: Taking advantage of operator overlap. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence, Detroit, Michigan, USA*, pages 949–953. Morgan Kaufman, San Mateo, 1989.

[HH89] J. Hertzberg and A. Horz. Towards a theory of conflict detection and resolution in nonlinear plans. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence, Detroit, Michigan, USA*, pages 937–943. Morgan Kaufman, San Mateo, 1989.

[HM86] S. Hanks and D.V. McDermott. Default reasoning, non-monotonic logic, and the frame problem. In *Proceedings of the 5th National Conference on Artificial Intelligence, Philadelphia, USA*, pages 328–333. Morgan Kaufman, Los Altos, 1986.

[Ind88] B. Indurkhya. Modes of analogy. In *Proceedings of the International Workshop on Analogical and Inductive Inference, Reinhardsbrunn Castle, GDR*, pages 217–229. Lecture Notes in Computer Science 397, Springer, Berlin, 1988.

[Kam89a] S. Kambhampati. Flexible reuse and modification in hierarchical planning: A validation structure based approach. PhD Thesis MD 207 42-3411, University of Maryland, Center for Automation Research, Computer Vision Laboratory, 1989.

[Kam89b] S. Kambhampati. Integrating planning and reuse: A framework for flexible plan reuse. In *Proceedings of the Workshop on Case-Based Reasoning, Pensacola Beach, Florida*, pages 280–284. Morgan Kaufman, San Mateo, 1989.

[Kam89c] S. Kambhampati. Representational requirements for plan reuse. In *Proceedings of the Workshop on Case-Based Reasoning, Pensacola Beach, Florida*, pages 20–23. Morgan Kaufman, San Mateo, 1989.

[Kam90a] S. Kambhampati. Mapping and retrieval during plan reuse: A validation structure based approach. In *Proceedings of the 8th National Conference on Artificial Intelligence, Boston, USA*, pages 170–175. MIT Press Menlo Park, Cambridge, London, 1990.

[Kam90b] S. Kambhampati. A theory of plan modification. In *Proceedings of the 8th National Conference on Artificial Intelligence, Boston, USA*, pages 176–182. MIT Press Menlo Park, Cambridge, London, 1990.

[Kau86] H.A. Kautz. The logic of persistence. In *Proceedings of the 5th National Conference on Artificial Intelligence, Philadelphia, USA*, pages 401–405. Morgan Kaufman, Los Altos, 1986.

[KC89] P. Koton and M. Chase. Knowledge representation in a case-based reasoning system: Defaults and exceptions. In *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning, Toronto, Ontario, Canada*, pages 203–211. Morgan Kaufman, Los Altos, 1989.

[Ker88] M. Kerber. Some aspects of analogy in mathematical reasoning. In *Proceedings of the International Workshop on Analogical and Inductive Inference, Reinhardsbrunn Castle, GDR*, pages 231–242. Lecture Notes in Computer Science, 397, Springer, Berlin, 1988.

[KH89] S. Kambhampati and J.A. Hendler. Control of refitting during plan reuse. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence, Detroit, Michigan, USA*, pages 943–949. Morgan Kaufman, San Mateo, 1989.

[KM90] H. Katsuno and A.O. Mendelzon. On the difference between updating a knowledge base and revising it. Technical Report KRR-TR-90-6, University of Toronto, 1990.

[Kno90] C.A. Knoblock. Learning abstraction hierarchies for problem solving. In *Proceedings of the 8th National Conference on Artificial Intelligence, Boston, USA*, pages 923–928. MIT Press Menlo Park, Cambridge, London, 1990.

[Kol84] J.L. Kolodner. *Retrieval and Organizational Strategies in Conceptual Memory - A Computer Model.* Lawrence Erlbaum, Hillsdale, New Jersey, 1984.

[Lei89] D. Leishman. Analogy as a constrained partial correspondence over conceptual graphs. In *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning, Toronto, Ontario, Canada*, pages 223–234. Morgan Kaufman, Los Altos, 1989.

[Lif87] V. Lifschitz. Formal theories of action. In *Readings in Nonmonotonic Reasoning*, pages 410–432. Morgan Kaufman, Palo Alto, 1987.

[Lin89] C. Lingenfelder. Structuring computer generated proofs. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence, Detroit, Michigan, USA*, pages 379–383. Morgan Kaufman, San Mateo, 1989.

[Lou89] R.P. Loui. Analogical reasoning, defeasible reasoning, and the reference class. In *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning, Toronto, Ontario, Canada*, pages 256–265. Morgan Kaufman, Los Altos, 1989.

[LR89] V. Lifschitz and A. Rabinov. Miracles in formal theories of action. *Artificial Intelligence*, 38:225–237, 1989.

[Luk88] W. Lukaszewicz. Chronological minimization of abnormality: Simple theories of action. In *Proceedings of the 7th Europeen Conference on Artificial Intelligence, Munich, Germany*, pages 574–576. Pitman, London, 1988.

[Lur88] M. Luria. Knowledge intensive planning. Technical Report UCB/CSD 88/433, Computer Science Division, University of California, Berkeley, 1988.

[M$^+$89] S. Minton et al. Explanation-based learning: A problem solving perspective. *Artificial Intelligence*, 40:63–118, 1989.

[MB86] R.J. Mooney and S.W. Bennett. A domain independent explanation-based generalizer. In *Proceedings of the 5th National Conference on Artificial Intelligence, Philadelphia, USA*, pages 551–555. Morgan Kaufman, Los Altos, 1986.

[McC77] J. McCarthy. Epistemological problems of artificial intelligence. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence, Cambridge, Massachusetts, USA*, pages 1038–1044. Morgan Kaufman, Los Altos, California, 1977.

[MH69] J. McCarthy and P.J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. American Elsevier, New York, 1969.

[Mun81] J.C. Munyer. *Analogy as a Means of Discovery in Problem Solving and Learning.* PhD thesis, University of California at Santa Cruz, 1981.

[MW86] Z. Manna and R. Waldinger. *How to Clear a Block: Plan Formation in Situational Logic.* Lecture Notes in Computer Science 230, Springer, Berlin, 1986.

[MW87] Z. Manna and R. Waldinger. A theory of plans. In *Reasoning about Actions & Plans, Proceedings of the 1986 Workshop*, pages 11–45. Morgan Kaufman, Palo Alto, 1987.

[Neb90] B. Nebel. *Reasoning and Revision in Hybrid Representation Systems.* Lecture Notes in Computer Science 422, Springer Verlag, Berlin, 1990.

[Red90] M. Redmond. Distributed cases for case-based reasoning; facilitating use of multiple cases. In *Proceedings of the 8th National Conference on Artificial Intelligence, Boston, USA*, pages 304–309. MIT Press Menlo Park, Cambridge, London, 1990.

[Rei80] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[RK89] D. Ruby and D. Kibler. Learning subgoal sequences for planning. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence, Detroit, Michigan, USA*, pages 609–615. Morgan Kaufman, San Mateo, 1989.

[RS89] C.K. Riesbeck and R.C. Schank. *Inside Case-based Reasoning.* Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1989.

[Sch82] R. Schank. *Dynamic memory: A Theory of Learning in Computers and People.* Cambridge University Press, 1982.

[Sha76] G. Shafer. *A Mathematical Theory of Evidence.* Princeton University Press, Princeton, New Jersey, 1976.

[Sha90] S.C. Shapiro, editor. *Enzyclopedia of Artificial Intelligence*, volume I and II. John Wiley and Sons, 1990.

[SHK89] R. H. Stottler, A.L. Henke, and J.A. King. Rapid retrieval algorithms for case-based reasoning. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence, Detroit, Michigan, USA*, pages 233–237. Morgan Kaufman, San Mateo, 1989.

[Sho86] Y. Shoham. Chronological ignorance: Time, nonmonotonicity, necessity and causal theories. In *Proceedings of the 5th National Conference on Artificial Intelligence, Philadelphia, USA*, pages 389–393. Morgan Kaufman, Los Altos, 1986.

[Sie89] J. Siekmann. Universal unification. *Journal of Symbolic Computation*, 7:207–274, 1989.

[Sim88] R.G. Simmons. A theory of debugging plans and interpretations. In *Proceedings of the 7th National Conference on Artificial Intelligence, Saint Paul, Minnesota, USA*, pages 94–99. Morgan Kaufman, San Mateo, 1988.

[SM90] E. Simoudis and J. Miller. Validated retrieval in case-based reasoning. In *Proceedings of the 8th National Conference on Artificial Intelligence, Boston, USA*, pages 310–315. MIT Press Menlo Park, Cambridge, London, 1990.

[Ten86] J. Tenenberg. Planning with abstraction. In *Proceedings of the 5th National Conference on Artificial Intelligence, Philadelphia, USA*, pages 76–80. Morgan Kaufman, Los Altos, 1986.

[Thi86] H. Thiele. A model theoretic oriented approach to analogy. In *Proceedings of the International Workshop on Analogical and Inductive Inference, Wendisch-Rietz, GDR*, pages 196–208. Lecture Notes in Computer Science 265, Springer, Berlin, 1986.

[Tho88] C.J. Thornton. Analogical inference as generalised inductive inference. In *Proceedings of the International Workshop on Analogical and Inductive Inference, Reinhardsbrunn Castle, GDR*, pages 254–263. Lecture Notes in Computer Science 397, Springer, Berlin, 1988.

[UR89] A. Unruh and P.S. Rosenbloom. Abstraction in problem solving and learning. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence, Detroit, Michigan, USA*, pages 681–687. Morgan Kaufman, San Mateo, 1989.

[WBH89] W. Wahlster, S. Biundo, and M. Hecking. PHI - Plan Based Help systems. Project description, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, 1989.

[Wil83] R. Wilensky. *Planning and Understanding.* Advanced Book Program. Addison-Wesley, Reading, Massachusetts, 1983.

RR-90-16
Franz Baader, Werner Nutt: Adding Homomorphisms to Commutative/Monoidal Theories, or: How Algebra Can Help in Equational Unification
25 pages

RR-91-01
Franz Baader, Hans-Jürgen Bürckert, Bernhard Nebel, Werner Nutt, and Gert Smolka :
On the Expressivity of Feature Logics with Negation, Functional Uncertainty, and Sort Equations
20 pages

RR-91-02
Francesco Donini, Bernhard Hollunder, Maurizio Lenzerini, Alberto Marchetti Spaccamela, Daniele Nardi, Werner Nutt:
The Complexity of Existential Quantification in Concept Languages
22 pages

RR-91-03
B.Hollunder, Franz Baader: Qualifying Number Restrictions in Concept Languages
20 pages

RR-91-05
Wolfgang Wahlster, Elisabeth André, Winfried Graf, Thomas Rist: Designing Illustrated Texts: How Language Production is Influenced by Graphics Generation.
17 pages

RR-91-06
Elisabeth André, Thomas Rist: Synthesizing Illustrated Documents
A Plan-Based Approach
11 pages

RR-91-07
Günter Neumann, Wolfgang Finkler: A Head-Driven Approach to Incremental and Parallel Generation of Syntactic Structures
13 pages

RR-91-08
Wolfgang Wahlster, Elisabeth André, Som Bandyopadhyay, Winfried Graf, Thomas Rist
WIP: The Coordinated Generation of Multimodal Presentations from a Common Representation
23 pages

DFKI Technical Memos

TM-89-01
Susan Holbach-Weber: Connectionist Models and Figurative Speech
27 pages

TM-90-01
Som Bandyopadhyay: Towards an Understanding of Coherence in Multimodal Discourse
18 pages

TM-90-02
Jay C. Weber: The Myth of Domain-Independent Persistence
18 pages

TM-90-03
Franz Baader, Bernhard Hollunder: KRIS: Knowledge Representation and Inference System -System Description-
15 pages

TM-90-04
Franz Baader, Hans-Jürgen Bürckert, Jochen Heinsohn, Bernhard Hollunder, Jürgen Müller, Bernhard Nebel, Werner Nutt, Hans-Jürgen Profitlich: Terminological Knowledge Representation: A Proposal for a Terminological Logic
7 pages

TM-91-01
Jana Köhler
Approaches to the Reuse of Plan Schemata in Planning Formalisms
52 pages

TM-91-02
Knut Hinkelmann
Bidirectional Reasoning of Horn Clause Programs: Transformation and Compilation
20 pages

DFKI Documents

D-89-01
Michael H. Malburg, Rainer Bleisinger: HYPERBIS: ein betriebliches Hypermedia-Informationssystem
43 Seiten

D-90-01
DFKI Wissenschaftlich-Technischer Jahresbericht 1989
45 pages

D-90-02
Georg Seul: Logisches Programmieren mit Feature -Typen
107 Seiten

D-90-03
Ansgar Bernardi, Christoph Klauck, Ralf Legleitner: Abschlußbericht des Arbeitspaketes PROD
36 Seiten

Approaches to the Reuse of Plan Schemata
in Planning Formalisms

Jana Köhler

TM-91-01
Technical Memo