Flexible Reuse of Plans by Abstraction and Refinement

Ralph Bergmann and Wolfgang Wilke
E-mail: bergmann@informatik.uni-kl.de
Centre for Learning Systems and Applications
University of Kaiserslautern
Dept. of Computer Science
P.O. Box 3049, D-67653 Kaiserslautern, Germany

Abstract

In this paper we present an approach to the flexible reuse of planning cases by abstraction and refinement. A new abstraction methodology is described in detail and related algorithms for automatically building and refining abstract planning cases are sketched. Based on a given concrete and abstract language together with a generic abstraction theory, this abstraction approach allows to change the whole representation of a planning case from a concrete to abstract.

1 Introduction

In this paper we present an approach to the flexible reuse of planning cases by abstraction and refinement. This approach is mainly inspired by the observation that reuse of plans must not be restricted to a single level of description. In domains in which the problems to be solved differ a lot, the reuse of past solutions must be achieved at various levels of abstractions. If past solutions that can be reused directly at the most detailed level of a description are not available, it is however possible to reuse solutions at an abstract level. An appropriate abstract solution can then be transferred to a solution to the new problem. Therefore, this abstract solution must be refined to become a complete solution at the required concrete level.

As the main contribution of this paper, we present a new abstraction methodology, a related algorithm for automatically building abstract planning cases from given concrete planning cases, and an algorithm for plan refinement. Based on a given concrete and abstract language together with a generic abstraction theory, this abstraction approach allows to change the whole representation of a planning case from a concrete to abstract. Abstract cases learned from several concrete cases are organized in a case-base for efficient retrieval during novel problem solving.

This approach is realized in PARIS (Plan Abstraction and Refinement in an Integrated System), which is fully implemented and tested in the domain of process planning in mechanical engineering. Figure 1 shows an overview of the whole system and its components. Besides case abstraction and refinement, PARIS also includes an explanation-based approach for generalizing cases during learning and for specializing them during problem solving (see [Bergmann, 1992a] for details). Furthermore, the system also includes additional mechanisms for evaluating different abstract and generalized cases according to computational effort required for matching, specialization, and refinement of cases. Based on this evaluation, different efficient indexing and retrieval mechanisms have been developed (see [Bergmann and Wilke, 1993] for details).

2 A Formal Model of Case Abstraction

In this section we present a new formal model of case abstraction which allows to change the representation language of a case from concrete to abstract. We use a STRIPS-like representation [Fikes and Nilsson, 1971] of operators and states to represent problem solving domains.

A problem solving domain \( D = \langle L, E, O, R \rangle \) is described by a language \( L \), a set of essential atomic sentences \( L_{\text{f}} \) of \( L \), a set of operators \( O \) with related descriptions, and additionally, a set of Horn clauses \( R \) out of \( L \). A state \( s \in S \) describes the dynamic part of a situation in a domain and consists of a
finite subset of ground instances of essential sentences of \( E \). With the symbol \( S \), we denote the set of all possible states descriptions in a domain, which is defined as \( S = \mathcal{E}^* \), with \( \mathcal{E}^* = \{ e \sigma \in \mathcal{E} \text{ and } \sigma \text{ is a substitution such that } e \sigma \text{ is ground} \}. In addition, the Horn clauses \( R \) allow to represent static properties which are true in all situations. An operator \( \delta(x_1, \ldots, x_n) \in \mathcal{O} \) is described by a triple \((Pre_\delta, Add_\delta, Del_\delta)\), where the precondition \( Pre_\delta \) is a conjunction of atoms of \( \mathcal{L} \), and the add-list \( Add_\delta \) and the delete-list \( Del_\delta \) are finite sets of (possibly instantiated) essential sentences of \( \mathcal{E} \). We write \( s_1 \mapsto s_2 \) to denote that the operator \( \delta \) transforms a state \( s_1 \) into a state \( s_2 \).

In addition to the concrete problem solving domain \( D_c = (\mathcal{L}_c, \mathcal{E}_c, \mathcal{O}_c, \mathcal{R}_c) \) we assume that an abstract problem solving domain \( D_a = (\mathcal{L}_a, \mathcal{E}_a, \mathcal{O}_a, \mathcal{R}_a) \) is additionally supplied by a domain expert. In the remainder of this paper states and operators from the concrete domain are denoted by \( s^c \) and \( \delta^c \), respectively, while states and operators from the abstract domain are denoted by \( s^a \) and \( \delta^a \) respectively.

The problem of case abstraction can now be described as transforming a case which consists of a problem description and a correct solution (linear plan) from the concrete domain \( D_c \) into a case in the abstract domain \( D_a \) (see Figure 2). Because the abstract language can be explicitly stated by the user, this abstraction methodology allows to change the representation language of cases (problems and solutions) completely from concrete to abstract. Therefore, a user can define the level of abstraction and the required representation language on which reuse of solutions is most beneficial for a domain.

By the abstraction transformation, the level of detail of the description of the case is reduced in two dimensions: at first, the level of detail of the description of individual states is reduced and second, the set of states that are kept in the abstract case is reduced too. Formally, this transformation is decomposed into two independent mappings: a state abstraction mapping \( \alpha \), and a sequence abstraction mapping \( \beta \) [Bergmann, 1992b; Bergmann and Wilke, 1995b; Bergmann and Wilke, 1995a]. The state abstraction mapping transforms a selection of concrete state descriptions that occur in the solution to a problem into abstract state descriptions, while the sequence abstraction mapping specifies which of the concrete states are mapped and which are skipped.

**Definition 1 (State Abstraction Mapping)** A state abstraction mapping \( \alpha : \mathcal{S}_c \to \mathcal{S}_a \) is a mapping from \( \mathcal{S}_c \), the set of all states in the concrete domain, to \( \mathcal{S}_a \), the set of all states in the abstract domain. In particular, \( \alpha \) must be an effective total function. This general definition of a state abstraction mapping does not impose any restrictions on the kind of abstraction besides the fact that the mapping must be a total many-to-one function. However, to restrict the set of all possible state abstractions to a set of abstractions which a user considers useful, we assume that additional domain knowledge about how an abstract state relates to a concrete state can be provided. This knowledge must be expressed in terms of a domain specific generic abstraction theory \( \mathcal{A} \).

**Definition 2 (Generic Abstraction Theory)** A generic abstraction theory is a set of Horn clauses of the form \( e_\sigma \leftrightarrow a_1, \ldots, a_n \). In these rules \( e_\sigma \) is an abstract essential sentence, i.e., \( e_\sigma = E_\sigma \sigma \) for \( E_\sigma \in \mathcal{E}_a \) and a substitution \( \sigma \). The body of a generic abstraction rule consists of a set of sentences from the concrete or abstract language, i.e., \( a_i \) are atoms out of \( \mathcal{L}_c \cup \mathcal{L}_a \).

Based on such a generic abstraction theory, we can restrict the set of all possible state abstraction mappings to those which are deductively justified by the generic abstraction theory.

**Definition 3 (Deductively Justified Mapping)** A state abstraction mapping \( \alpha \) is deductively justified by a generic abstraction theory \( \mathcal{A} \), if the following conditions hold for all \( s^c \in \mathcal{S}_c \):

- if \( \phi \in \alpha(s^c) \) then \( s^c \cup \mathcal{R}_c \cup \mathcal{A} \vdash \phi \) and
- if \( \phi \in \alpha(s^c) \) then for all \( s^c \) such that \( s^c \cup \mathcal{R}_c \cup \mathcal{A} \vdash \phi \) holds, \( \phi \in \alpha(s^c) \) is also fulfilled.

In this definition the first conditions assures that every abstract sentence reached by the mapping is justified by the abstraction theory. Additionally, the second requirement guarantees that if an abstract sentence is used to describe an abstraction of one state, it must also be used to describe the abstraction of all other states, if the abstract sentence can be derived by the generic abstraction theory.

For the selection of those concrete states that have a corresponding abstraction, the sequence abstraction mapping is defined as follows:

**Definition 4 (Sequence Abstraction Mapping)** A sequence abstraction mapping \( \beta : \mathbb{N} \to \mathbb{N} \) relates an abstract state sequence \( (s^a_0, \ldots, s^a_n) \) to a concrete state sequence \( (s^c_0, \ldots, s^c_n) \) by mapping the indices \( j \in \{1, \ldots, m\} \) of the abstract states \( s^a_j \) into the indices \( i \in \{1, \ldots, n\} \) of the concrete states \( s^c_i \), such that the following properties hold:

- \( \beta(0) = 0 \) and \( \beta(n) = n \): The initial state and the goal state of the abstract sequence must correspond to the initial and goal state of the respective concrete state sequence.
- \( \beta(u) < \beta(v) \) if and only if \( u < v \): The order of the states defined through the concrete state sequence must be maintained for the abstract state sequence.

Based on the two abstraction functions introduced, our intuition of case abstraction is captured in the following definition.
Definition 5 (Case Abstraction) A case \( C_a = (\langle s_0^a, s_m^a \rangle, (\alpha_1^a, \ldots, \alpha_n^a) \rangle \) is an abstraction of a case \( C_c = (\langle s_0^c, s_m^c \rangle, (\alpha_1^c, \ldots, \alpha_n^c) \rangle \) with respect to the domain descriptions \( \langle D_a, D_b \rangle \) if \( s_i^c \xrightarrow{\alpha_j} s_i^c \) for all \( i \in \{1, \ldots, n\} \) and \( s_j^c \xrightarrow{\alpha_j} s_j^c \) for all \( j \in \{1, \ldots, m\} \) and if there exists a state abstraction mapping \( \alpha \) and a sequence abstraction mapping \( \beta \), such that: \( s_j^c = \alpha(s_j^a) \) holds for all \( j \in \{0, \ldots, m\} \).

This definition of case abstraction is demonstrated in Figure 2. The concrete space shows the sequence of \( n \) operations together with the resulting state sequence. Selected states are mapped by \( \alpha \) into states of the abstract space. The mapping \( \beta \) maps the indices of the abstract states back to the corresponding concrete states. In [Bergmann and Wilke, 1995b] we showed that in this abstraction methodology hierarchies of abstraction spaces as well as different kinds abstraction can be handled simultaneously. Moreover, our abstraction methodology is more powerful than abstraction by dropping conditions as often realized in hierarchical planning [Sacerdoti, 1974; Knoblock, 1994].

![Figure 2: General idea of case abstraction](image)

3 Computing Case Abstractions

We have developed an algorithm for automatically learning a set of abstract cases from a given concrete case. Thereby, we assume that the concrete domain \( D_c \) and the abstract domain \( D_a \) are given together with the generic abstraction theory. Please note that we do not require any kind operator abstraction rules.

Roughly speaking, the algorithm consists of four separate phases. In the first phase, the sequence of concrete states which results from the execution of the concrete solution is computed. The second phase derives for each concrete state all possible abstract essential sentences justified by the generic abstraction theory. In the subsequent phase, a graph of all applicable abstract operators is constructed, in which each edge leads from an abstract state to an abstract successor state. Finally, all consistent paths, starting at the abstract initial state and leading to the final abstract state are determined. Each of these paths represents a case which is an abstraction of the concrete case.

In [Bergmann and Wilke, 1995a] we have described and analyzed this abstraction algorithm in detail and showed that it is sound and complete with respect to our abstraction methodology.

4 Retrieving and Refining Abstract Cases

Now we assume a case-base which contains a set of abstract cases that can be reused to solve a new problem. During problem solving, an abstract case must be selected from a case-base, and the abstract plan contained in this case must be refined to become a solution to the new problem. During case retrieval we must search for an abstract case which is applicable, i.e., it contains a problem description that is an abstraction of the current problem. In order to decide whether an abstract problem description is an abstraction of the current problem in hand, we apply the generic abstraction theory already used during case abstraction. Thereby an abstract problem description of the new problem is inferred from the concrete problem description. The abstracted problem description can be matched against the abstract problem description contained in the abstract case.

If a matching case is found, it must be refined to come to a solution to the current problem. This refinement starts with the concrete initial state from the problem statement. A search is performed to find a sequence of concrete operations which leads to a concrete state that can be abstracted with the generic abstraction theory to match the second abstract state contained in the abstract case. If the first abstract operator can be refined and abstracted with the generic abstraction theory to match the second abstract state, we have a valid abstract plan. If the refinement fails, we can backtrack to the refinement of the previous operator and try to find an alternative refinement. If the whole refinement process reaches the final abstract state, the abstract plan is complete.

This refinement demands for a search procedure which allows an abstract goal specification. All kinds of forward-directed search such as depth-first iterative-deepening or best-first search procedures can be used for this purpose. In PARIS we use depth-first iterative-deepening search.

Please note that PARIS allows the reuse of problem decompositions at different levels of abstraction. Abstract plans decompose the original problem into a set of much smaller subproblems. These subproblems are solved by a search-based problem solver. The problem decomposition leads to a significant reduction of the overall search that must be performed to solve the problem [Korf, 1987]. With pure search the worst-case time complexity for finding the required solution by search is \( O(b^n) \), where \( n \) is the length of the solution and \( b \) is the
5 An Example Domain

This section presents the example domain we have selected from the field of process planning in mechanical engineering. We have selected the goal of generating a process plan for the production of a rotary-symmetric workpiece on a lathe. The problem description contains the complete specification (especially the geometry) of the desired workpiece (goal state) together with a specification of the piece of raw material (called mold) it has to be produced from (initial state). A chucking fixture, together with the attached mold, is rotated with the longitudinal axis of the mold as rotation center. As the mold is rotated a cutting tool moves along some contour and thereby removes certain parts of the mold until the desired goal workpiece is produced. Within this process it is very hard to determine the sequence in which the specific parts of the workpiece have to be removed and the cutting tools to be used. In this domain, the problems to be solved and the resulting solutions differ very much at the level of the representation on which they are typically specified. This motivates the use of a more flexible reuse approach that allows to change the representation language.

In Figure 3 we show how an example case can be abstracted and how this abstract case can then be reused to solve a different planning problem. The top of this figure shows a concrete planning case \( C_1 \). The representation of this case contains the exact geometrical specification of each element of the contour. Several areas of this contour are named by the indicated coordinates (e.g. \( #2, #2 \)) for further reference in the cut-operations of the plan. The resulting abstract case is shown in the center of the figure. The abstract solution plan consists of a sequence of 6 abstract operators. The sequence of the operators in the plan is indicated by the Roman numerals. The particular abstraction is indicated between the concrete and the abstract case and denotes which sequence of concrete operators is turned into which abstract operator. The learned abstract case can now be used to solve the new problem \( C_2 \) whose initial and final concrete states are shown in the bottom of the figure. Even if the concrete workpiece looks quite different from the workpiece in case \( C_1 \) the abstract case can be used to solve the problem. We can see that most abstract operators (in particular the operators II, VI, and V) are refined to completely different sequences of concrete operators than those from which they were abstracted. This demonstrates the flexibility of this approach to plan reuse.

In [Bergmann and Wilke, 1995b] we also report on a series of different experiments designed for evaluating PARIS. Table 1 summarizes some of the experimental results that were obtained by using PARIS to solve 100 different process planning problems. In this experiment, PARIS was trained with 5 different cases (10 cases in the second run) which are available for reuse during problem solving. The table summarizes the percentage of problems that could be solved and the average computation time required to solve each problem. The results are compared to those obtained in a separate experiment in which each problem is solved by pure search without reuse.

<table>
<thead>
<tr>
<th>Reuse Cases</th>
<th>Solved Problems</th>
<th>Solution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>reusing 5 cases</td>
<td>83 %</td>
<td>50 sec.</td>
</tr>
<tr>
<td>reusing 10 cases</td>
<td>86 %</td>
<td>56 sec.</td>
</tr>
<tr>
<td>no reuse</td>
<td>79 %</td>
<td>157 sec.</td>
</tr>
</tbody>
</table>

Table 1: Percentage of solved problems and average solution time.

6 Requirements and Limitations

The most important prerequisite of this method is the availability of the required background knowledge, namely the concrete world description, the abstract world description, and the generic abstraction theory. For the construction of a planning system, the concrete world description must be acquired anyway, since it specifies the language of the problem description (essential sentences) and the problem operators (operators).
abstract world and the generic abstraction theory must be acquired additionally. We feel that this is indeed the price we have to pay to make reuse more flexible and thereby planning more tractable in certain practical situations.

Nevertheless, the formulation of an adequate abstract domain theory is crucial to the success of the approach. If those abstract operators are missing which are required to express a useful abstract plan, reuse cannot be achieved. What we need are mostly independently refinable abstract operators. If such operators exist, they can be simply represented in the abstract domain using the whole representational power. Additionally, the requirement that the abstract domain is given by the user has also the advantage that the learned abstract cases are expressed in terms the user is familiar with. Thereby, the user can understand an abstract case very easily. This can open up the additional opportunity to involve the user in the planning process, for example in the selection of an abstract case she/he favors.

7 Related Work

Most similar to the PARIS approach are the case-based planning systems Prodigy/Analogy [Veloso and Carbonell, 1993] and PRIAR [Kambhampati and Hendler, 1992]. However, these systems reuse planning cases directly and without doing any abstraction. Knoblock [Knoblock, 1994] presented an approach to automatically constructing abstraction hierarchies. This approach is limited to abstraction by dropping conditions and does not allow to change the representation language. It also does not allow to reuse cases. Our approach is also related to the idea of skeletal plans [Friedland and Iwasaki, 1985]. In the skeletal plan approach no model of the operators (neither concrete nor abstract) is used to describe the preconditions and effects of operators as is done in PARIS. There is no explicit notion of states and abstraction or refinement of states. Instead, the plan refinement is achieved by stepping down a hierarchy of operators, guided by heuristic rules for operator selection.

References


