Learning Hierarchically Clustered Shared Plan Abstractions as Problem Solving Knowledge with High Utility for Planning

Ralph Bergmann
University of Kaiserslautern
W-6750 Kaiserslautern, Germany
E-Mail: bergmann@informatik.uni-kl.de

Abstract
Complex problem solving can be substantially improved by the reuse of experience from previously solved problems. This requires that case libraries of successful problem solutions are transformed into problem solving knowledge with high utility, i.e., knowledge which causes high savings in search time, high application probability and low matching costs. Planning can be improved by explanation-based learning (EBL) of abstract plans from detailed, successfully solved planning problems. Abstract plans, expressed in well-established terms of the domain, serve as useful problem decompositions which can drastically reduce the planning complexity. Abstractions which are valid for a class of planning cases rather than for a single case, ensure a successful application in a larger spectrum of new situations. The hierarchical organization of the learned shared abstractions causes low matching costs. The presented S-PABS procedure is a combination of EBL and incremental conceptual clustering approaches to automatically construct a hierarchy of shared abstract plans by analyzing concrete planning cases.

1 Introduction
Recently, a lot of approaches have been proposed to improve planning by using machine learning techniques. Many approaches favor the analysis of success cases of problem solutions to create new amounts of knowledge which can be additionally used by a planning system. Planning improvement is usually achieved through the shortening of the problem solving process by the reuse of earlier problem solving experience. While in case-based reasoning [Kolodner, 1987] a large collection of very detailed cases is organized for efficient retrieval and modification, traditional learning approaches prefer the extraction of more general knowledge from specific problem solving cases. With explanation-based learning techniques (EBL) [Mitchell et al., 1986] background knowledge about a problem domain can be incorporated to extract control rules [Minton et al., 1989], macro-operators [Fikes et al., 1972; Korf, 1985; Tadepalli, 1991], or skeletal plans [Bergmann, 1992b] as generalizations of successful planning cases. Unfortunately, it shows that simply storing generalizations (e.g., as a list of macro-operators) does not guarantee a speed-up effect in general. This is because the sequential search for applicable generalizations and their matching usually exceeds the savings caused by their utilization. This major problem of EBL is called the utility problem [Minton, 1990]. The utility of a learned generalization (rule) can be estimated by the following cost/benefit formula also used in the PRODIGY system [Minton et al., 1989]:

\[ Utility = (AvrSavings \times ApplieFreq) - AvrMatchCost \]

where \( AvrSavings \) is the average time savings produced when the generalization is applicable due to the search elimination, \( ApplieFreq \) is the probability that the rule is applicable, and \( AvrMatchCost \) is the average time cost of matching the rule.

In the following we want to examine techniques which can be integratively combined to a new procedure which analyses previously solved planning cases to learn problem solving knowledge with high utility for new planning problems. Rules with a high utility are consequently those rules which lead to high savings (\( AvrSavings \)) in search time, high application probability (\( ApplieFreq \)) and low matching costs (\( AvrMatchCost \)).

A lot of research on planning has intensively examined the savings in search caused by abstraction [Sacerdoti, 1974; Friedland and Iwasaki, 1985]. The main computational advantage of having a good
abstract solution to a planning problem — an abstract plan — is that one large search space of the complexity \( b^n \) (where \( b \) is the branching factor and \( n \) the length of a solution plan) can be decomposed into \( k \) subproblems with smaller search spaces, in which the complete problem solution requires a reduced complexity of \( b^{n_1} + b^{n_2} + \ldots + b^{n_k} \) \( (n_1 + n_2 + \ldots + n_k = n) \). As already identified by Knoblock [Knoblock, 1989] it is important to avoid backtracking across subproblems. Fairly independently solvable subproblems should be aspired for which domain specific abstraction knowledge is highly demanded. So, learning domain tailored abstractions of successful problem solutions instead of generalizations seems promising since abstractions have shown to lead to significant savings in search time.

The second factor which influences the utility is the application probability of the learned rules. Rules should be applicable for a large spectrum of problems to be solved in the future. Future problems are usually unknown, but can assumed to be similar to the available cases of previous problem solutions. With that assumption, we can aim at learning rules which are applicable for a large set of the previous problems. This leads to approaches for explanation-based learning from multiple examples [Flann and Dietterich, 1989].

To reduce the matching costs, Yoo and Fisher [Yoo and Fisher, 1991] have recently proposed the construction of a hierarchical classification tree as memory organization structure to store generalizations of examples. A rule stored at a node in such a hierarchy is always more general than the rules stored at any of it’s successor nodes. When the match of a rule in the hierarchy fails, all successor nodes can consequently be discarded. So the overall matching costs for the rules can be significantly reduced.

From the combination of the central ideas of these three approaches — abstraction, learning from multiple examples, and hierarchical classification — a learning procedure can be constructed to efficiently improve planning performance through past problem solving experience. The requirements for such a learning procedure can be summarized as follows:

1. **maximize AvrSavings**: A learning procedure should learn from success cases and should construct abstract plans which serve as useful decompositions of a planning problem into several smaller problems. The created abstractions should be tailored to the application domain and should use an established terminology which has shown to lead to independently solvable subproblems in human problem solving.

2. **maximize ApplicFreq**: Problem decompositions should be learned that are shared by larger class of planning cases rather than only by a single case. Thereby, abstractions can be selected which can promise to be successfully applicable in a larger spectrum of new situations.

3. **minimize AvrMatchCost**: The body of the learned knowledge must be arranged in a memory organization structure (hierarchical classification tree) which allows an efficient retrieval of the abstract plans.

The rest of this paper presents an approach to automatically transforming a given set of successful planning cases into a hierarchy of abstract plans, where each abstract plan is a shared abstraction which is valid for a whole set of previous planning problems rather than for a single problem. The next section formally introduces the meaning of shared abstractions of plans. In section three, the five phase S-PABS (Shared Plan Abstraction) procedure is presented as a method which learns from a set of plans and comes out with shared abstract plans. Section four shows how methods of incremental concept formation can be utilized to construct a classification tree of the learned abstractions. The described approach is demonstrated for the familiar 'Towers of Hanoi' domain. Finally, section five discusses how the required background knowledge can be acquired for real world domains and related work is presented in connection with the S-PABS approach.

### 2 Formal Model of Plan Abstraction

Michalski and Kodratoff [Michalski and Y. Kodratoff, 1990] have recently pointed out that abstraction has to be distinguished from generalization. While generalization transforms a description along a set-superset dimension, abstraction transforms a description along a level-of-detail dimension which usually involves a change in the representation space from the original into a simpler language [Plaisted, 1981; Tenenberg, 1987; Giordana et al., 1991; Mozetic and Holzbaur, 1991]. A plan is composed of operations which are executed in a specific order and thereby successively change the state of a world. Within this
planning model, abstraction has two independent dimensions: On the first dimension a change in the level-of-detail for the representation of single states is described. On the second dimension a change in the level-of-detail is declared by reducing the number of states contained in a plan. As a consequence, a change of the representation of the state description and a change of the operations which describe the state transitions is required. Both dimensions of abstraction are essential to achieve a reduction of the complexity for planning.

2.1 Abstraction of a Single Plan

In the following, the formal model of plan abstraction from a single plan [Bergmann, 1992] is introduced. The commonly used notation for the description of worlds, states and plans (e.g. [Fikes et al., 1972; Lifschitz, 1987; Knoblock, 1989]) is further assumed:

Definition 1 (STRIPS-World) A STRIPS-world $W$ is a triple $(R, T, Op)$ over a first-order language $L$, where $R$ is a set of essential sentences [Lifschitz, 1987] which describe the dynamic aspects of a state of the world. $T$ is a static theory which allows to deduce additional properties of a state in the world. $Op$ is a set of operators represented by descriptions $(Pa, Da, Ao)_{a \in Op}$, where $Pa$ is the precondition formula, $Da$ is the delete list and $Ao$ is the add list [Fikes et al., 1972].

As usual, a state $s$ of a world $W$ is described by a subset of the essential sentences from $R$. The theory $T$ is implicitly assumed to be valid in all states of the world. Let $S = 2^R$ be the set of all states of the world.

A plan $p$ in a world $W$ is a sequence $(o_1, \ldots, o_n)$ of operators from $Op$. In a world $W$ an initial state $s_0 \in S$ and a plan $p = (o_1, \ldots, o_n)$ induce a sequence of states $s_1 \in S, \ldots, s_n \in S$ where $s_{i-1} \cup T \vdash Pa_i$ and $s_i = (s_{i-1} \setminus Da_i) \cup Ao_i$.

Two plans $p = (o_1, \ldots, o_n)$ and $p' = (o'_1, \ldots, o'_n)$ are called equivalent iff in every state $s_i = s'_i \in S$ follows that $s_i = s'_i, i \in \{1, \ldots, n\}$ for the states induced by the plans.

In the following, we assume that the two world descriptions $W_c = (R_c, T_c, Op_c)$ (the concrete world) and $W_a = (R_a, T_a, Op_a)$ (the abstract world) are given, as background knowledge of appropriate descriptions of the planning domain. The problem of plan abstraction can now be described as transforming a plan $pa$ from the concrete world $W_c$ into a plan $pc$ in the abstract world $W_a$, with several conditions being satisfied. In the presented model, this transformation is formally decomposed into two mappings, a state abstraction mapping $a$, and a sequence abstraction mapping $b$ as follows:

Definition 2 (State Abstraction Mapping) A state abstraction mapping $a: S_c \rightarrow S_a$ is a mapping from $S_c$, the set of all states in the concrete world, to $S_a$, the set of all states in the abstract world, that satisfies the following conditions:

a) If $sc \cup T_c$ is consistent then $a(sc) \cup T_a$ is consistent for all $sc \in S_c$ (maintain consistency).

b) If $sc \cup sc' \cup T_c$ is consistent then $a(sc \cup sc') \subseteq a(sc) \cup a(sc')$ for all $sc, sc' \in S_c$ (monotony property).

The state abstraction mapping transforms a concrete state description into an abstract state description and thereby changes the representation of a state from concrete to abstract. The consistency of states is maintained to exclude abstractions from which arbitrary consequences can be drawn. Furthermore we require the monotony property for state abstraction mappings to avoid that already computed abstractions become invalid if new concrete properties are added.

Definition 3 (Sequence Abstraction Mapping) A sequence abstraction mapping $b: N \rightarrow N$ relates an abstract state sequence $(sa_0, \ldots, sa_m)$ to a concrete state sequence $(sc_0, \ldots, sc_n)$ by mapping the indices $i \in \{1, \ldots, n\}$ of the abstract states $sa_i$ into the indices $j \in \{1, \ldots, m\}$ of the concrete states $sc_j$, so that the following properties hold:

a) $b(0) = 0$ and $b(n) = m$: 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.

b) $b(u) < b(v)$ iff $u < v$: The order of the states defined through the concrete state sequence must be maintained for the abstract state sequence.
Abstract Space: \[ \text{Concrete Space:} \]

State abstraction mapping: \[ s(0) \rightarrow a(0), s(1) \rightarrow a(1), \ldots, s(n) \rightarrow a(n) \]

Concrete Space: \[ \text{Sequence abstraction mapping: } b(0) = 0, b(1) = 2, \ldots, b(n) = m \]

Figure 1: Demonstration of the Plan Abstraction Methodology

The sequence abstraction mapping specifies which of the concrete states are mapped onto an abstract state. This mapping selects the concrete states with a relevant abstract interpretation. Note, that the initial and the goal state must always get abstracted.

**Definition 4 (Abstraction)** A plan \( p_a \) is an abstraction of a plan \( p_c \) if there exists a state abstraction mapping \( a: S_c \rightarrow S_a \) and a sequence abstraction mapping \( b: N \rightarrow N \), so that: If \( p_c \) and an initial state \( s_{c0} \) induce the state sequence \( (s_{c0}, \ldots, s_{cm}) \) and \( s_{a0} = a(s_{c0}) \) and \( (s_{a0}, \ldots, s_{an}) \) is the state sequence which is induced by \( s_{a0} \) and the abstract plan \( p_a \), then \( a(s_{c(i)}) = s_{a(i)} \) holds for all \( i \in \{1, \ldots, n\} \).

This definition of abstraction is demonstrated by Figure 1. The concrete space shows the sequence of \( m \) operations together with the induced state sequence. Selected states induced by the concrete plan (i.e., \( s_{c0}, s_{c2}, \) and \( s_{cm} \)) are mapped by the state abstraction mapping \( a \) into states of the abstract space. The sequence abstraction mapping \( b \) maps the indices of the abstract states to the corresponding states in the concrete world. It becomes clear, that plan abstraction is defined by a pair of abstraction mappings \((a, b)\). This is because these mappings uniquely define an equivalence class of abstract plans according to the defined plan equivalence (if their exists an abstract plan at all). Note, that if a concrete and an abstract world is given, many different not equivalent plans can be an abstraction for the same concrete plan according to definition 4.

### 2.2 Domain Justification for State Abstraction Mappings

To reduce the number of all possible abstraction to those which are potentially useful, a justification of the state abstraction mappings by knowledge of the domain is necessary. Therefore, additional background knowledge should state which kinds of useful abstractions usually occur in a domain, and how those abstractions can be defined in concrete terms. Generic abstraction theories for semantic abstraction, as introduced by Giordana, Rovere and Saitta [Giordana et al., 1991] relate atomic formulae of an abstract language to terms of a corresponding concrete language. In an adaptation of this idea, a generic state abstraction theory \( T_\Psi \) in our model is defined as a set of axioms of the form \( \Psi \leftarrow D_1 \vee D_2 \vee \ldots \vee D_n \), where \( \Psi \) is an essential sentence of the abstract world and \( D_1, \ldots, D_n \) are conjunctions of sentences of the concrete world. For a justified state abstraction mapping \( a \) we can now require that: if \( \Psi \in a(s_c) \) then \( s_c \cup T_c \cup T_\Psi \vdash \Psi \). Since a minimal consistent state abstraction mapping (according to the \( \vdash \) relation) should be reached, the reverse implication, namely that every essential sentences \( \Psi \) for which \( s_c \cup T_c \cup T_\Psi \vdash \Psi \) holds, is an element of \( a(s_c) \), is not demanded.

### 2.3 Shared Plan Abstractions

The definition of plan abstraction according to definition 4 only describes the abstraction of a single plan. An abstraction which is shared by a larger class of concrete solution plans and therefor represents a problem decomposition which is applicable for several problem cases, has a higher application probability (ApplicFreq) for new problems and is consequently of higher utility. Therefore, the formalization of abstraction must be extended to handle shared abstractions, which are valid for a set of plans.

Intuitively, an abstract plan which holds for a set of plans \( P \) must be an abstraction for each of the plans. So, for each of the plans to be abstracted, a state abstraction mapping and a sequence abstraction...
mapping must be found, so that the same sequences of abstracted state descriptions are created. This idea is captured in the next definition.

**Definition 5 (Shared Abstractions)** Let $P = \{\langle p_{c1}, s_{c0,1} \rangle, \ldots, \langle p_{ck}, s_{c0,k} \rangle\}$ be a set of $k$ plans $p_{ci}$ and belonging initial states $s_{cij}$. Let $(s_{c0,n}, s_{c1,n}, \ldots, s_{cm,n}) = a(\ldots, v)$ denote the sequences of states which are induced by $P$. A plan $p_{sa}$ is a shared abstraction of a set of plans $P$ if there exist $k$ state abstraction mappings $a(\ldots, v)$ and $k$ sequence abstraction mappings $b(\ldots, v)$, so that: If $s_{0a} := a(1, \ldots, a(k))$ and $(s_{a0}, \ldots, s_{ak})$ is the state sequence which is induced by $s_{0a}$ and the shared abstract plan $p_{sa}$, then $a(1, \ldots, a(k))$ is a set of all shared abstraction plans for $P$, we define $SA(P) := \{s_{a} | p_{sa}$ is shared abstraction of $P\}$ as the set of all abstract plans for $P$.

From this definition it is easy to see, that the following inclusion holds: $SA(P2) \subseteq SA(P1)$ if $P1 \subseteq P2$. This means that enlarging the set of concrete plans from $P1$ to $P2$ reduces the set of shared abstractions. If the set of concrete plans grows to large, it can even happen that no shared abstraction exists anymore for the whole set.

### 2.4 An Example

To illustrate the formal model of shared plan abstraction a small example is introduced now. As a planning domain, the familiar Tower-of-Hanoi (ToH) problem — a problem analyzed by several researchers before (e.g., Korf, 1985) — is used. The ToH-problem involves three vertical pegs and a number of shaped disks, all of different sizes. In the initial state, all the disks are stacked on one peg in decreasing order of size. The goal is to stack the disks in the same order on one of the other pegs. The only legal action is to move a single top disk on a peg to another peg subject to the constraint, that a larger disk may never be placed on a smaller disk.

Figure 2 shows two example problem solutions to the 2-disk and the 3-disk ToH problem. In the top of this figure, the plan $p_{c1}$ for the 2-disk ToH-problem is shown. In the initial state $s_{c0,1}$ the two disks are stacked on the left most peg $a$. The other pegs $b$ and $c$ are empty. The stacked disks of the three pegs are represented as three columns of natural numbers, where the value of a number reflects the sizes of the respective disks (small numbers stand for small disks). An empty peg is represented by the underscore (_) symbol. The first operation of the plan $p_{c1}$ is the move of the top disk (1) on peg $a$ to peg $b$.
b. This operation is represented by the term \( m(a, b) \), which denotes the application of the operator *move* (abbreviated by \( m \)) with the two parameters source peg \( a \) and destination peg \( b \). The second operation moves the top disk from peg \( a \) to peg \( c \) and the third operation achieves the goal state in which all pegs are correctly stored on the right most peg \( c \). The bottom of Figure 2 shows the plan \( pc_3 \) as a solution to the 3-disk ToH-problem. The three disks (numbered 1, 2, 3) are moved with 7 legal moves from peg \( a \) to peg \( c \). The shared abstraction which is derived according to the model of plan abstraction is indicated between the two concrete plans.

Here, the abstract world consists of a different terminology for the descriptions of the states and operations than the concrete world. In the state descriptions, the symbol \( fis \) introduced as an abstraction of a complete tower of all disks contained in a problem. The symbol \( f \) stands for the largest disk and \( s \) abbreviates a small tower of disks, which is a tower that does not contain the largest disk. The generic abstraction theory which is required to allow justified abstraction mappings exactly defines these new abstract objects \((t, l, s)\) in terms of the concrete disks. Additionally, new abstract operations which act on the changed state representation are introduced. The operation *split* splits a complete tower \( f \) into the largest disk and the small remaining tower \( s \). The operation \( m' \) moves a single disk or any tower to a new location. The *join* operation stacks a small tower on top of a large disk. Note that all these abstract operations are not legal elementary operations of the concrete ToH-domain. The two state abstraction mappings and the two sequence abstraction mappings which define the shared abstraction of the 2-disk and the 3-disk problem are listed in Table 1. \( a^{(1)} \) maps the concrete from the plan \( pc_1 \) onto the abstract states and \( a^{(2)} \) maps the concrete states which result from the plan \( pc_2 \) onto the same sequence of abstract states. The sequence abstraction mapping \( b^{(1)} \) mirrors the fact, that each state in the solution of the 2-disk problem is mapped onto a respective abstract state, while for the 3-disk problem only 4 of the 8 states have an abstract image according to \( b^{(2)} \).

3. **Explanation-based Learning of Abstractions**

As described in section 2, the task of constructing a shared plan abstraction can be seen as the problem of finding deductively justified state abstraction mappings and sequence abstraction mappings so that abstract plans — composed of the operators of the abstract world — exists. This section introduces S-PABS\(^3\) (shared plan abstraction), an EBL-procedure which computes \( SA(P) \) from the set of concrete plans \( P \), using the given concrete and abstract world descriptions as well as the generic abstraction theory as background knowledge. This procedure consists of five distinct phases in which the different types of background knowledge are applied to infer explanations for all concrete plans from which shared abstract plan can be derived. The first three phases are executed separately for each of the plans in \( P \). Phase-IV superimposes the candidate abstract explanations yielding a set of shared plans, which are further variabilized in phase-V. In the following the five phases are explained in detail.

3.1 **Phase-I: Plan Simulation (Application of Concrete World Knowledge)**

By simulating the execution of the concrete plan \( pc \), the state sequence \((sc_1, \ldots, sc_m)\) which is induced by the plan \( pc \) and a given initial state \( sc_0 \) is computed (see Figure 1). During this simulation, the definition of the operators \( Op_c \) and the static theory \( T_c \) are applied to derive all those essential sentences which holds in the respective states. The proofs that exist for the applicability of each operator can now be seen as a concrete-level explanation for the effects caused by the operations. Such a kind of explanation is also constructed in EBL-procedures for plan generalizations [Fikes et al., 1972; Chien, 1989; Bergmann, 1992b; Bergmann, 1992a]. For the 3-disk ToH-problem the computed state sequence \((sc_{0,2}, \ldots, sc_{7,2})\) is shown in the lower section of Figure 2.

3.2 **Phase-II: Constructing State Abstractions (Application of the Generic Abstraction Theory)**

The second phase performs a prerequisite for the composition of the deductively justified state abstraction mapping. With the generic state abstraction theory \( T_g \), an abstract state description \( sa'_2 \) is derived for

\(^2\)To simplify the presentation, the shown state abstraction mappings only note the mapping of disk configurations, although they formally map complete states, which are sets of essential sentences.

\(^3\)S-PABS is an extension of the PABs procedure, described in [Bergmann, 1992c].
each state $sc_i$ which was computed in the first step. Essential sentences $\Psi \in R_s$ of the abstract world description $W_s$ are checked, whether they can be inferred from $sc_i \cup T_c \cup T_g$. If $sc_i \cup T_c \cup T_g \vdash \Psi$ holds, then $\Psi$ is included into the state abstraction $sa_i^l$. Although, each of the concrete states is transformed with the guidance of the generic abstraction theory into abstract descriptions, not every state has a meaningful abstract interpretation. For which states the abstraction turns out to be useful, can ultimately be answered in phase-IV. For the 3-disk ToH-problem, some of the abstract essential sentences which are derived from the states $sc_{i,3}, sc_{i,2}, sc_{i,2}, sc_{i,2}$ are shown in the center of Figure 2 and also in Figure 3.

### 3.3 Phase-III: Constructing Abstract State Transitions (Application of Abstract World Knowledge)

The goal of the third phase is to identify candidate abstract operations for an abstract plan. For each pair of abstracted states $(sd_i, sa_i^l)$ with $u < v$, it is checked, if there exists an abstract operation $O_a \in OP_a$ described by $< P_a, D_a, A_a >$ which is applicable in $sd_i$ and which transforms $sd_i$ into $sd'_i$. If $sd_i \cup T_c \cup T_g \vdash P_a$ and if every sentence of $A_a$ is contained in $sd_i'$ and none of the sentences of $D_a$ is contained in $sd_i'$ then the operation $O_a$ is noted to be a candidate for the abstract plan. A directed graph is constructed, where the nodes of the graph are built by the abstract states $sa_i^l$ and where links between the states are introduced for those operations that are candidates for achieving the respective state transitions. The proofs that exist for the validation of $P_a$ in $sa_i^l$ are stored together with the corresponding operation. A fraction of the complete graph of the candidate abstract operations is shown Figure 3. For some of the abstract states, the description derived in phase-II is shown. In addition to the disk abstraction symbols $t, l, s$ which have already been introduced in section 2.4, the symbol $u$ represents the second largest disk and the symbol $v$ represents any tower (complete or partial) which does not contain the largest or the second largest disk. The presented graph can be derived from the 3-disk ToH-problem as well as for then 4-disk ToH-problem.

### 3.4 Phase-IV: Establishing Shared Consistent Paths

From the constructed graph, complete and consistent paths from the initial abstract state $sd_i^l$ to the final state $sd_m^l$ are searched, where each path determines a complete abstract explanation. The consistency requirement for such a path $pa = (v_1, \ldots, v_n)v_i \in OP_a$ expresses, that every essential sentence which guarantees the applicability of the operator $v_i$ is created by a preceding operation (through the add-link) and is protected until $v_{i+1}$ is applied, or the essential sentence is already true in the initial state and is protected until $v_{i+1}$ is applied. This condition assures that the plan represented by the path $pa$ is indeed applicable, which means that the preconditions for all operations are satisfied in the states in which they are executed. This consistency requirement can be verified by analyzing the dependencies of the involved operations. The graph shown in Figure 3 consists of five paths from the initial to the final abstract state, but only three paths, marked $< \alpha >, < \beta >$, and $< \gamma >$ fulfill the introduced consistency requirement. Although two states (and one operation) are shared by the paths $< \beta >$ and $< \gamma >$, the crossing of them does not represent a consistent path. This is because the operations in path $< \beta >$ work on the disk abstractions noted by the symbols $t, l, s$ whereas the operations in path $< \gamma >$ rely on a more detailed view on the disk-configuration represented by the symbols $l, n, u$.

The modules of the S-PABS procedure reported so far, does not take the construction of shared explanations into account. They are executed separately for each of the plans to learn from. The
determined set of consistent paths can then easily be superimposed to select only those paths which are shared by all of the example plans. For judging if some paths are shared, it is important to take the intermediate states induced by the plans into consideration too (refer to definition 5).

In the example graph from Figure 3, we can identify that the path \(< \alpha >\) is shared by the plans for the 1, 2, 3 and 4-disk ToH-problem. Path \(< \beta >\) represents an explanation for the 2, 3 and 4-disk problem, while path \(< \gamma >\) only holds for the 3 and the 4-disk problem. From this example we can also see, that possibly more then one path can survive the process of intersection. So all three paths \(< \alpha >, < \beta >,\) and \(< \gamma >\) are shared consistent paths for the 3 and the 4-disk ToH-problem. In this case S-PABS will come out with several plan abstractions from which some may be selected for further usage.

### 3.5 Phase-V: Constructing the Final Abstract Plan Representation

From a shared abstract path and the dependency network which justifies its consistency, a variabilization of the abstracted plans can be established. With the dependency network, which functions as an explanation structure, explanation-based generalization can be applied to compute the least subexplanation which justifies all operations of the abstract path. The proofs that correspond to the justification of the abstract states by the generic abstraction theory are pruned. Within the resulting subexplanation, the remaining derivations are generalized by standard goal regression as used by Mitchell, Keller and Kedar-Cabelli [Mitchell et al., 1986]. Thereby, constants are turned into variables. The final generalized explanation thus only contains relations which describe the generalized operations together with a generalized specification of the application conditions for the operator sequence. As an example, the variabilized abstract plan which results from path \(< \beta >\) is shown in Figure 4. Note that the capital letters X, Z, Y now indicate variables which stand for the pegs of ToH.

### 4 Classification of Plan Abstractions

This section deals with approaches to an efficient organization and utilization of abstract plans learned by the S-PABS procedure to keep the overall matching costs low. As already proposed by Yoo and Fisher [Yoo and Fisher, 1991], concept formation over explanations is a method that combines the explanation-based paradigm with the paradigm of concept formation [Gennari et al., 1989] to result in a method which can automatically create a hierarchical classification tree of shared explanations.

#### 4.1 Fundamentals for the Hierarchy Construction

The basic idea is to construct a classification hierarchy, in which each node in the hierarchy reflects the shared abstraction of a set of planning cases. The abstract plan stored at a node is valid for all of the node’s descendents. A descendent of a node represents a more specific abstract plan, which, when applicable, causes a smaller search space than all of the node predecessors. If an abstract plan at a node is not applicable, than all of the node’s descendents will also not be applicable and need not to be visited.

To construct such a classification hierarchy according to the above mentioned requirements, we remember the following property of shared abstractions: \( \mathcal{S}A(P1) \subseteq \mathcal{S}A(P2) \) if \( P2 \subseteq P1 \). This statement expresses the obvious property, that abstractions learned from a set of concrete plans are also valid abstractions for any subset. On the other hand, it is clear that extending the set of plans from which abstractions are to be constructed may reduce the set of resulting shared abstractions. A classification of abstract plans can be build on the basis of a classification of the concrete planning cases. If \( C_i \) is a node in the classification hierarchy, then let \( EC_i \) denote the set of the concrete example plans which
Figure 5: Classification Hierarchy of Plan Abstractions

belong to that class. If \( C_i \) is a subclass of \( C_j \) in the hierarchy, then \( EC_i \subseteq EC_j \) holds and therefore \( \mathcal{SA}(EC_j) \subseteq \mathcal{SA}(EC_i) \) is also true. An abstract plan \( pa_{C_i} \) must be associated with each class \( C_i \), where \( pa_{C_i} \in \mathcal{SA}(EC_i) \). A typical representative abstract plan for a class \( C_i \) should be chosen in a way that it differs from the abstract plans which have been selected for the superclasses of \( C_i \). To achieve this condition, \( pa_{C_i} \) can be designated as follows: \( pa_{C_i} \in (\mathcal{SA}(EC_i) \setminus \mathcal{SA}(EC_j)) \) if \( C_j \) is superclass of \( C_i \). The application domain as well as the classification hierarchy has an important impact on the size of the space of candidates (\( \mathcal{SA}(EC_i) \setminus \mathcal{SA}(EC_j) \)) from which to select an abstract plan for a class. A perfect hierarchy should be constructed in a way, that for each class the space of candidates contains only one item. In this case the set of abstract plans which is constructed for a set of example plans \( EC_i \) is distributed over all the nodes along the path from the root node of the hierarchy to the class \( C_i \). An example of such a classification hierarchy for the ToH domain is demonstrated in Figure 5, where 8 different problems are classified. As the first four problems (noted as 1, 2, 3, 4) the traditional 1, 2, 3 and 4-disk ToH-problems (see section 2.4) are concerned. The other four problems 2*, 3*, 4*, 5* represent a variation of the typical ToH-problems. These problems deal with the merging of two fragmented towers on the pegs \( a \) and \( b \) into one complete tower located on peg \( a \). The 4-disk merge problem (4*) and its concrete solution is briefly illustrated in Figure 6. In the hierarchy shown in Figure 5, the abstract plans for the different classes and the set of problems from which they are derived by S-PABS are shown. This hierarchy is constructed so, that each class can be characterized by a sole plan since \( |\mathcal{SA}(EC_i) \setminus \mathcal{SA}(EC_j)| = 1 \). Only the root node of the hierarchy does not contain an abstract plan because no shared abstractions between the traditional ToH-problems and the merge problems exists.

4.2 Incremental Construction of the Hierarchy

For an incremental construction of such a classification hierarchy, a newly observed solution to a planning problem \( E \) must be incorporated into an existing hierarchy. For that purpose, S-PABS computes \( \mathcal{SA}(EC_j) \), \( \mathcal{SA}((E)) \) and \( \mathcal{SA}(EC_i \cup \{E\}) \) for a class \( C_i \) of the hierarchy (initially the root). When the three sets of resulting abstractions are compared, three different situations are distinguished:

a) If \( \mathcal{SA}(EC_i \cup \{E\}) = \mathcal{SA}(EC_j) = \mathcal{SA}((E)) \) then the new example is simply discarded and not incorporated, because all of the examples abstractions are already contained in a class of the hierarchy.
b) If it shows that $\mathcal{SA}(EC_i \cup \{E\}) = \mathcal{SA}(EC_i)$ but $\mathcal{SA}(EC_i) \neq \mathcal{SA}(\{E\})$ then $E$ becomes a member of the class $C_i$ and the classification proceeds to the descendants of $C_i$. One subclass $C_j$ is chosen in which $E$ fits best. This selection is guided by a criterion that can be rated by $|\mathcal{SA}(EC_i) - |\mathcal{SA}(EC_i \cup \{E\})|]$, the number of abstractions of the class $C_j$ which are not shared by $E$. (If no subclass can be chosen according to this criterion, then the subclass which contains the largest number of abstract plans is selected). If $C_j$ is a leaf node of the hierarchy, then a new subclass $C_i$ is created, which exactly contains the example plan $E$.

c) If $\mathcal{SA}(EC_i \cup \{E\}) \neq \mathcal{SA}(EC_i)$ then the new example does not completely fall in the scope class $C_i$ (which is the best selection as guaranteed in case b). Therefore, a new class node called $C_k$ is created and inserted between $C_i$ and its father. This new class is initialized with the example plans of $C_i$ supplemented by the example $E (EC_k := EC_i \cup \{E\})$. Additionally, if $\mathcal{SA}(EC_k) \neq \mathcal{SA}(\{E\})$ a new child of $C_k$ is created, which contains only the example plan $E$.

With this procedure the classification hierarchy shown in Figure 5 can be incrementally constructed from the eight Toll-problems.

5 Discussion

5.1 Acquisition of the Required Background Knowledge

In this paper, a method was described which analyzes previously solved planning cases to derive problem solving knowledge with high utility for improving planning processes. 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 descriptions must be acquired anyway, since they specify the 'language' of the problem description (essential sentences) and the problem solution (operators). The abstract world and the generic abstraction theory must be additionally acquired. This is indeed the price we have to pay to make planning more tractable. Other hierarchical planning approaches (e.g. [Stefik, 1981; Friedland and Iwasaki, 1985; Paulokat and Wess, 1993]) even require the acquisition of multi-level hierarchies of operators. Research on knowledge acquisition has shown that human experts used to employ lots of abstract knowledge to cope with the complexity of real-world planning problems, e.g. in program synthesis [Jeffries et al., 1988; Vorberg and Goebel, 1991] or production planning [Thoben and Schmalhofer, 1990]. Specific knowledge acquisition tools have been developed to comfortably acquire such abstract knowledge [Musen et al., 1987; Bergmann and Schmalhofer, 1991; Schmidt and Zickwolff, 1992] from different sources.

The specific knowledge needs of S-PABS could be fulfilled for the domain of program synthesis of sequential machine-level programs [Bergmann, 1992c]. In this domain, the concrete world specifies the semantics of the operations of a machine-level programming language. The abstract world represents programming constructs of a higher-level programming language and the generic abstraction theory specifies the abstract data types of the high-level language in terms of the available machine-level data.

5.2 Related Work

Within the Soar framework, Unruh and Rosenbloom [Unruh and Rosenbloom, 1989] have proposed an abstraction technique which can be characterized as general weak method, in that it uses no domain-specific knowledge about how to perform abstractions. This is in contrast to our approach, since we want to draw power from the knowledge about useful domain specific abstractions which have been proven successful in human problem solving.

Unlike other well known techniques for learning search control rules for planning (PRODIGY) by explanation-based learning [Minton et al., 1989], S-PABS can acquire domain oriented problem decompositions rather then more or less restricted operator selection rules. Search control rules can guide the search in a single problem space but cannot reduce planning complexity by switching to an abstract problem description. On the other hand PRODIGY is able to learn from failed solution tracks which actually cannot be performed by S-PABS.

Recently, a few approaches to plan abstraction have been proposed. In Knoblock's method for learning abstract planning spaces [Knoblock, 1989], abstraction always occurs by dropping sentences of the
concrete world. This kind of abstraction is only a special case of the type of abstractions we allow. This restrictions can be characterized by limiting state abstraction mappings those, where \( q(s) \subseteq s \) is satisfied. Plan abstractions how they are derived by PLANEREUS [Anderson and Farly, 1988] or by Tenenberg’s approach [Tenenberg, 1986] can also be shown to be a special type of the abstraction created by S-PABS with the specific sequence abstraction mapping \( b(i) := i \).

Similar to S-PABS, the CAbPlan-system [Paulokat and Wess, 1993] — a more traditional case-based reasoning system — also uses planning cases to guide the search of a hierarchical non-linear planner. But planning cases required by CAbPlan are supposed to contain solutions to the planning problem on all reasoning system also uses planning cases to guide the search of a hierarchical non-linear planner. But

Acknowledgements

I would like to thank Andreas Dannenmann, Dagmar Surmann, and Wolfgang Wilke and especially the reviewers Roman Cunis and E. Sommer for providing very helpful comments on early versions of this paper.

References


