An Integrated Planning Representation using Macros, Abstractions, and Cases

Jacky Baltes and Bruce MacDonald
2500 University Drive NW
Calgary, Alberta T2N 1N4, Canada
{baltes,bruce}@cpsc.ucalgary.ca

Abstract

Planning will be an essential part of future autonomous robots and integrated intelligent systems. This paper focuses on learning problem solving knowledge in planning systems. The system is based on a common representation for macros, abstractions, and cases. Therefore, it is able to exploit both classical and case–based techniques. The general operators in a successful plan derivation would be assessed for their potential usefulness, and some stored. The feasibility of this approach was studied through the implementation of a learning system for abstraction. New macros are motivated by trying to improve the operator–set. One heuristic used to improve the operator–set is generating operators with more general pre–conditions than existing ones. This heuristic leads naturally to abstraction hierarchies. This investigation showed promising results on the towers of Hanoi problem. The paper concludes by describing methods for learning other problem solving knowledge. This knowledge can be represented by allowing operators at different levels of abstraction in a refinement.

Introduction

This paper advocates a common representation for operators that includes abstract plans, cases and macros [Baltes, 1991]. An important aspect of this representation is that a system should be able to learn the necessary problem solving knowledge.

The implementation of a prototype system that learns abstraction hierarchies is described. The learning system tries to improve the operator–set by extracting macros with more general pre–conditions than existing ones. This leads naturally to the generation of abstraction hierarchies. Rather than searching for new macros explicitly, the learner extracts new macros from a successful plan. It tries to find operators that result in identical states and that differ in exactly one pre–condition predicate. If such operators are found, the system deletes the differing predicate from the pre–conditions, thus forming an abstract operator. Since the planning system is intended to support case–based planning techniques, a generalized and an instantiated version of the macro is stored. We intend to use a novel dynamic filtering scheme [Baltes, 1991] to delete poor macros.

The learning system was tested on the towers of Hanoi problem and showed promising results. The remainder of the paper is organized as follows: first, the paper presents a common representation for planners, then reviews previous work on operator learning with macros. Section explains how operators are learned in our representation. Then, a description of the implementation and an example are given. The paper concludes by describing how we intend to learn other problem solving knowledge such as reactive rules or anticipation of failure.

Macro–Operators

A linear macro is a sequence of primitive operators. This sequence is usually generalized and added to the operator set as a new operator. For example, useful macros in the blocks world domain are **pickup=(goto,grasp)** and **putdown=(goto,ungrasp)**. Macros can be used in the construction of new macros, for example **move=(pickup,putdown)**.

This paper focuses on linear macros because the formation of iterative or disjunctive macros depends on good linear ones [Shell and Carbonnel, 1989]. Macros speed up the planning process because they reduce the solution length. On the other hand, the generation of macros must be carefully controlled because new operators increase the branching factor of the search space. Minton showed that simply generating all possible macros from a successful solution decreases performance [Minton, 1985].

Dynamic Filters

As mentioned above, only a small number should be generated, ideally ones that will be useful in future problem solving tasks. The basic problem of macro learning is that the system has to predict the useful-

ness of a macro based on its previous experience. The following paragraphs describe the effect of adding one macro to the operator set and derive a *usefulness* measure for such an addition. This measure can be used to dynamically delete macros.

By adding a macro m, the branching factor is increased. However, the new macro will not be applicable in all situations. Let b be the branching factor without the macro in question. Let c be the fraction of states where m is applicable. Furthermore, not all applications of m will lead to a solution, so let u_m be the *usefulness* of m, which is the overall chance of applying m to achieve a solution; the ratio of the total number of times m leads to a solution, to the total number of times any operator is applicable. If l_m is the number of primitive operators in m, then the time complexity for the new operator set is of the order:

$$(b+c)^{n/l_m u_m}$$

The branching factor is increased by the applicability of m, and the effective solution length is reduced by the chances of using m, and in proportion to m's length. If u_m is 1, this means m is used at each step of the solution, and the plan is divided in length by the number of operators in m. If planning is to be faster when a macro is added, then the following inequality must be satisfied:

$$b^n \geq (b+c)^{n/l_m u_m} \quad (1)$$

$$\log b \geq \frac{1}{l_m u_m} \log(b+c) \quad (2)$$

$$u_m \geq \frac{1}{l_m} \cdot \frac{\log(b+c)}{\log b} \quad (3)$$

The macro length predominates the generality of its pre–conditions, c, in 3. So it will be more important to allow long macros than more specific macro pre–conditions. However, the pre–conditions cannot be ignored; impractically large values are required for b to make the second fraction in 3 approach unity. Note that u_m and l_m are not independent; as the length grows the chances of the macro being used in a plan decrease. Furthermore, it is also assumed that the space searched does not change with m. Under this assumption, b and c are independent.

Equation 3 cannot be used directly for selecting new macros because u_m, b, and c cannot be effectively computed *a priori*. However, these values can be approximated statistically. After a number of trials, equation 3 can be used as a dynamic filter to remove unnecessary macros.

Iba [Iba, 1989] proposes dynamic filters in his MACLEARN system. However, the implementation seems ad hoc; the user determines when to call the dynamic filter routine, which deletes all macros that have not been used at least once in a previous problem solving episode.

The utility measure is not based on the number of pre–conditions in a macro–operator (as it is done in Minton's system), since as will be explained in the remainder of the paper, the number of pre–conditions does not increase in my system.

Macros, Abstractions, and Cases

This section suggests a common representation for macros, abstractions, and cases in a planning system. It will show the similarity and differences between these methods, and suggest that a common representation will allow a problem solver to use all three strategies simultaneously.

The proposed representation will enable a planning system to maintain important advantages of previous systems:

- The planner will learn only when there is strong motivation, in order to increase performance in the future. This point has been shown by the MACLEARN system (flatten the search space, [Iba, 1989]) and by the CHEF system (repair failed plans and anticipate problems, [Hammond, 1989]).
- Proposed macros are filtered statically as well as dynamically. A new heuristic described in equation 3 is used.
- The planner learns from a worked example (similar to MACLEARN, PiL2 [Yamada and Tsuji, 1989], CHEF) rather than using a brute force search to find new operators (which would be similar to MPS ,[Korf, 1985].
- The planner should be able to use a heuristic function or other knowledge that is available.

Korf mentioned the similarity between abstractions and macros [Korf, 1987]. Both methods try to reduce the search by generating a skeleton search space of the original problem space. Instead of searching in the original space, a solution is found in the skeleton space and this solution is then refined into a solution in the original problem space. One difference, however, is that there can be more than one abstraction level whereas macros normally generate only one skeleton space.

Cases can be viewed as long, specific macros. The main distinction between macros and cases is the way in which they are used in a planning system. Cases are fetched from memory and some plan critics are applied to change the case to the new situation. Macros are usually not altered, i.e. the sequence of elementary operators is not adapted to the new situation.

The common feature among all three items is that the most important information stored is a set of pre–conditions and a set of effects, as for elementary operators.

Abstraction hierarchies are equivalent to elementary operators that are missing some pre–condition predicates. This means that although the specific execution depends on all pre–conditions, the effects can be achieved independently of the actual value of the

deleted predicates in the pre–conditions. One abstract operator can be specialized in a number of different ways. The representation can capture this by associating a set of operators with pre–conditions and effects. This structure represents that the effects can be achieved given that the pre–conditions are true, but that the instantiation of the plan may depend on predicates not mentioned in the pre–conditions. A method similar to PiL2's perfect causality heuristic is used to generate new operators that depend on fewer pre–conditions. More than one level of abstraction can be represented by showing that elements of the refinement of an abstract operator can consist of abstract operators.

Common representation

In our common representation, shown in Figure 1, an operator is recursively represented as (a) a pre–conditions and effects pair, and (b) a set of refinements, each of which is an operator sequence. A primitive operator has no refinements, and can be executed.

A variety of well–known problem solving knowledge is supported. An operator is like a macro if (a) there is only one refinement, (b) each operator in the refinement is a primitive, and (c) pre–conditions and effects predicate arguments are instantiated in neither the operator nor the refinement (i.e. the macro has formal parameters). A case is an operator with a refinement that is a fully instantiated (long) sequences of primitives (i.e. an instantiated macro). An abstract operator at criticality level k has refinement/s whose operators are abstract ones at level $k-1$. This representation supports *relaxed* (predicates deleted from pre–conditions, ABTWEAK) as well as *reduced* (predicates deleted in pre–conditions and effects, ALPINE) models of abstraction [Knoblock, 1991].

An operator is a subgoal sequence if all refinements contain no primitive operators (e.g. means ends analysis). Anticipation of failure can be represented by an operator whose single refinement is a single operator pre–conditions, effects pair in which there is an additional effect (such as "avoid soggy broccoli"). This ensures that the planner knows about the problem, and the refinement of the failure anticipation operator will be expanded using the successful plan, which is also stored as an operator. Since our representation does not enforce a common level of abstraction for operators in the refinement, a case or macro can also be generalized by making some operators non–primitives. This allows us to store adaptations of a case such as the replacement of some steps. Reactive rules may be represented as an operator whose single, two–operator refinement is a fully instantiated, primitive first operator, followed by a non–primitive pre–conditions, effects pair. If this two–operator refinement is reversed, then the resulting operator is suitable for backward chaining from the goal (similar to RWM [Güvenir and Ernst, 1990]).

While this generality provides a common operator representation, it also presents the immediate problem of controlling the creation of operators, so that planning is not impossibly expensive. We intend to control this using the dynamic operator deletion mechanism introduced above. In addition, the learning methods that add operators to the case memory must do so only when there is strong justification, and must choose "important" parts of new plans for storage, deciding the level of abstraction, number of refinements, and so on. This is the subject of current work. The common representation enables us to treat the various kinds of planning system in a single consistent framework, to better aid analysis and comparison.

The planner may choose to "forget" the refinements of some operators, when their usefulness decreases. But the pre–conditions, effects pair is retained, and the details can be replanned if necessary.

Planning using a common Representation

This section indicates how one might use the operator representation given in this paper. The planner should combine case–based as well as classical planning techniques, to take advantage of both previous experience, and the ability to solve new problems. What is needed is a control strategy that recalls and uses previous experiences to solve similar new problems, but gracefully moves into classical planning if no similar cases can be found. The recursive structure of the representation lends itself well to a recursive control strategy. Learning is designed to support and improve the planning process, by storing new operators. The planner should restrict the branching factor of the search space by focusing on a small number of operators instead of all applicable ones.

The input to the planner are initial state, goal state, and the operator set. Additional input is a resource limit and a skeleton plan agenda, which may support resource limited and multiple task planning. The planning begins by matching and recalling stored operators that have pre–conditions and effects similar to the current state and goal. The refinement/s of these operator/s will give various types of plans to be considered for solving the current problem.

Recalling similar operators A stored, similar case may have additional pre–conditions or effects, or be missing some. Operators should be recalled when their pre–conditions and/or effects are similar to the current goal and initial state. Possible indexing schemes for recall can be based on the number of predicates in pre–conditions and effects, the predicates themselves, or combinations of predicates.

Recalling is independent of the learned operator hierarchy; the fetched operator is not necessarily at the top level of a refinement tree. For example, if there is an abstract operator to move a medium disk independently of the small disk, and one refinement of this is

KEY: [Pre, Post] indicates a non-primitive operator, < Pre, Post > a primitive one, and <[Pre, Post]> either. $<[Pre, Post]>^k$ is an operator with predicates at criticality level k or above.

General Operator:

$$[Pre, Post] \longrightarrow \begin{array}{l} <[Pre_{11}, Post_{11}]>, <[Pre_{12}, Post_{12}]> \ldots <[Pre_{1n_1}, Post_{1n_1}]> \\ <[Pre_{21}, Post_{21}]>, <[Pre_{22}, Post_{22}]> \ldots <[Pre_{2n_2}, Post_{2n_2}]> \\ \vdots \\ <[Pre_{m1}, Post_{m1}]>, <[Pre_{m2}, Post_{m2}]> \ldots <[Pre_{mn_m}, Post_{mn_m}]> \end{array}$$

Each $Pre_{i1} \Rightarrow Pre$ and each $Post_{in_i} \Rightarrow Post$.

Macro (uninstantiated arguments) or Case (instantiated arguments):

$$[Pre, Post] \longrightarrow < Pre, Post_1 >< Pre_2, Post_2 > \ldots < Pre_n, Post >$$

Abstract operator:

$$[Pre, Post]^k \longrightarrow \begin{array}{l} <[Pre_{11}, Post_{11}]>^{k-1}, <[Pre_{12}, Post_{12}]>^{k-1} \ldots <[Pre_{1n_1}, Post_{1n_1}]>^{k-1} \\ <[Pre_{21}, Post_{21}]>^{k-1}, <[Pre_{22}, Post_{22}]>^{k-1} \ldots <[Pre_{2n_2}, Post_{2n_2}]>^{k-1} \\ \vdots \\ <[Pre_{m1}, Post_{m1}]>^{k-1}, <[Pre_{m2}, Post_{m2}]>^{k-1} \ldots <[Pre_{mn_m}, Post_{mn_m}]>^{k-1} \end{array}$$

Automatic subgoaling: $[Pre, Post] \longrightarrow [Pre, Post_1][Pre_2, Post_2] \ldots [Pre_n, Post]$

Anticipation of failure: $[Pre, Post] \longrightarrow [Pre, Post_1]$

Reactive rules: $[Pre, Post] \longrightarrow < Pre, Post_1 >[Pre_2, Post]$

RWM-type operator subgoal: $[Pre, Post] \longrightarrow [Pre, Post_1]< Pre_2, Post >$

Figure 1: The representation of planning operators.

to move the medium disk when both are on the first peg, and if the current state is that both disks are on that peg, that refinement is retrieved, rather than a more abstract one.

Adapting an existing plan Adaptation of a plan to new initial states and goal states is done by analysing the differences among the initial state and the pre–conditions and among the goal state and the effects of the similar plan. There are a number of general purpose adaptations to a plan that substitute one operator, listed below. If these don't provide a complete plan, then we treat the adapted plan as a subplan and use means ends analysis to complete it.

Replace Steps: An operator should be removed and steps inserted to achieve either pre–conditions or effects.

- Remove Side effect: If a plan fails because one operator has a specific side effect try to replace this operator with one that works.
- Protect effect: A following operator destroys an effect of the solution. Try to replace this operator with one that does not change the side effect.

Substitution: Replace a variable instantiation with a different object (the operator sequence is unchanged).

To find where to replace an operator, pre–conditions of the case that are not given in the current situation are pushed forward up to the first operator depending on those pre–conditions. Then the planner finds all elements of the effects that are dependent on this operator. If the effects are also part of the current goal, the system generates a new planning problem from the state just before the operator with the non–matching pre–condition to the first operator that uses any of the effects established. For example, if the goal is to have a barbecue and one of the pre–conditions is to have a match, this pre–condition is pushed forward to the operator **make-fire**. Since fire is a prerequisite for a barbecue, this effects is pushed backward to the operator **put steaks on fire** which has **has-fire** as a pre–condition. The system then tries to "improvise" and generate a plan using the state just before the operator **make-fire** to operator **put steaks on fire**. Given that we have a lighter in the current state, this plan can be easily generated. The **light match** operator is replaced by the **use lighter** operator. "Remove side effects" and "Protect effect" are specializations of the Replace operator strategy.

If the non–matching pre–condition does not establish a current goal predicate, the system tries to apply all operators of the plan, substituting variables where necessary (e.g. beans for broccoli). For example, given a plan to make a beef and bean dish from the ingredients, and if the system returns a plan for a beef and broccoli dish, the pre–condition have broccoli does not establish any predicate in the current goal (beef and

beans dish). In this case, the system simulates the plan and uses beans instead of broccoli.

After the case has been fixed to achieve all its effects with the new initial conditions, the system tries to achieve missing goal conditions one by one, using means ends analysis. The first non–satisfied term of the goal conjunction is selected and a new planning problem is generated from the goal state of the case to the goal state of the original problem.

The planner computes the subgoals that are necessary for the achievement of any of the adaptations or classical planning rules. It then retrieves similar cases for each of the generated subgoals and tries to work on them in order of similarity. This can also be used to repair failed plans, if the failed plan is stored in memory with a new effects added so that the failure is avoided.

Learning General Operators

Many researches have investigated different methods for constructing macros [Korf, 1985; Korf, 1987; Minton, 1985; Iba, 1989; Yamada and Tsuji, 1989]. A comparison of those methods leads to the following issues:

Generalized macros Korf's MPS system [Korf, 1985] stores instantiated macros, whereas Yamada's PiL2 system [Yamada and Tsuji, 1989] and Iba's MAC–LEARN system [Iba, 1989] generalize macros, so that they are more widely applicable. Although generalization of macros seems intuitive, it also increases the search space, especially if many objects exist in the domain.

Worked examples The MPS system searches for macros to fill the table. In the worst case, this may prevent the algorithm from terminating, although a solution to the problem may exist. This can occur if MPS is trying to find an impossible macro. PiL2 and MAC–LEARN use a "worked example" to extract macros. A " worked example" is either a successful plan or part of the search space that was searched when trying to find a successful plan. This means that no extra search effort is required for the generation of macros.

Motivation The motivation behind the MPS system is to combine automatic subgoaling with macros. Macros are used to *serialize* a subgoal sequence by guaranteeing that the goal conditions of previous subgoals are satisfied after the application of the macro, although they may be temporarily destroyed during its application. The heuristic used in the MAC–LEARN system is to generate macros between peaks in the heuristic evaluation function. This means that macros are used to flatten the search space of the heuristic evaluation function so that valleys can be traversed faster. The PiL2 system uses the *perfect causality* heuristic. It extracts a macro from a successful plan if (a) the pre–conditions of an operator in the plan were not satisfied in the initial state, and (b) the pre–conditions of this operator were satisfied after the application of previous operators. The motivation is to generate macros that allow the system to apply more operators to the initial state.

Learning Abstraction Hierarchies with Macros

Although a common representation is powerful, the manual generation of useful operators requires a large amount of domain knowledge and is tedious. Ideally, the planning system should learn operators from its previous experience. Therefore, a learning system was designed to create new operators for the representation. There are two major motivations for the system to learn:

Failure The system generates a plan that failed. Here, it tries to avoid generation of invalid plans in the future. Examples are anticipation of failure in case–based planning systems or explanation based learning rules.

Success Given that the system generated a successful plan, extract information from this plan to speed up the process for similar goals in the future. The generation of macro–operators and automatic subgoaling fall into this category.

The "need to learn" is easily recognized in the failure driven approach. The system knows exactly when new information has to be added, that is exactly when a generated plan failed. The problem is to decide what information should be stored in order to avoid failure in the future. For example, should the fully instantiated problem be stored or a generalization of it.

Learning in the success driven approach is harder, because the system must decide when to integrate new knowledge as well as what knowledge to integrate. For example, the MAC–LEARN system motivates learning by trying to flatten the search space. In PiL2, a sequence of operators that are used only to generate pre–conditions of a following operator should be combined in one macro so that the operator can be applied to the initial state.

This paper proposes two new heuristics for the generation of macros. The motivation is that to improve performance, a macro learner must improve the operator set. A macro–learner only changes the operator set, it does not tell the planner when to apply new operators. For example, it does not affect the heuristic evaluation function. Previous systems such as MAC–LEARN and PiL2, however, do not take the current operator set into consideration when learning new macros. There are two ways in which an operator can be improved.

Heuristic 1 Try to create new abstract operators that contain fewer pre–conditions than existing operators, and identical effects. That means that certain conditions can more easily be generated.

Heuristic 2 Try to introduce operators that have fewer effects than existing ones. This generates operators with more specific effects, so that the planning system can affect the world more controlled.

Heuristic 1 is more interesting because it generates an abstraction hierarchy of operators. This paper describes the implementation of a macro–learner that uses only the first heuristic to find new macros.

Implementation of the Macro–Learner

The learning system described in this paper is an addition to the AbTweak planning system implemented by Yang [Yang and Tenenberg, 1990]. The macro–learner generates a successful plan using AbTweak and extracts macros from it.

Figure 2 is a pseudo code description of the algorithm used. Explanation based generalization (EBG) is a common technique for the generalization of a macro [Minton, 1985]. The problem is given a sequence of operators and variable instantiations to compute the weakest set of pre–conditions that still allow the achievement of its effects.

Post–Conditions The post–conditions of an operator are the set of facts that must be true after application of the operator. It is different from the effects of an operator because the effects only mention facts that are *changed* by the operator. However, the pre–conditions that are not affected must also be true after application of the operator. The post–conditions are equivalent to the effects plus all predicates in the pre–conditions that are unchanged.

Logically Equivalent Descriptions The major problem in the implementation of the system is that there is more than one possible logical description of the world. For example, in the towers of Hanoi problem, the states `(not ons Peg1)(not ons Peg2)` and `(ons Peg3)` are equivalent because there are only three possible pegs and each disk must always be on a peg. The macro–learner extracts macros that have the same post–conditions but can be used to generate abstract macros. This means that the algorithm must establish the logical equivalence of world states. There are two possible solutions to this problem.

The first method uses a resolution theorem prover to prove the equivalence of post–conditions. This method is the most general one. A set of axioms can be given that can be used to prove facts about the domain. Since the operator set describes all elementary actions by which the world can be affected, it is also possible to derive certain facts used in the proof. For example, since the only operator that moves the small disk establishes the fact that the small disk is on some peg, the system can derive the fact that the disk is always on some peg.

The second method uses a unique description of the world. Two states are identical if and only if they have the same description. This can be achieved by changing the representation of operators or by designing a set of domain dependent rewrite rules that change a description dynamically. For example, a rewrite rule can be used to convert `(not ons Peg1)(not ons Peg2)` to `(ons Peg3)`.

This projects focuses on the feasibility of learning abstract operators rather than the design of a practical planning system. Therefore, the standard description language of operators was changed to generate a unique description of all world states. For example, the standard definition of the operator to move the big disk in the towers of Hanoi problem is

```
move-big($X $Z)
    Pre: (onb $X) (not ons $X) (not ons $Z)
         (is-peg $X) (is-peg $Z)
   Post: (onb $Z) (not onb $X)
```

This definition was changed to allow unique descriptions of post–conditions. Therefore, the `move-big` operator was defined as follows:

```
move-big($X $Y $Z)
    Pre: (onb $X) (ons $Y)
         (is-peg $X) (is-peg $Y) (is-peg $Z)
   Post: (onb $Z) (not onb $X)
```

This means that all operators must reference all three pegs in their argument list, and that all facts are represented directly instead of indirectly.

Example

This section shows an example of the macro–learner in the towers of Hanoi domain with two disks. There are three reasons for selecting this problem.

- It was easy to find a representation of operators that resulted in a unique logical description of the world.
- The problem is well studied and comparison to other planners can be made. Also, the optimal solution for this problem is known.
- The structure of the problem is well suited to abstract operators. In fact, abstract operators can reduce its time complexity to be linear in the length of the solution.

In the initial state, both disks are on the first peg. The goal is to move both disks on the third peg. The standard operator set is changed to use a unique logical description and is represented by:

```
Macro-Learner(Plan, Operator-Set)
   Compute all possible macros in the plan.
   For each macro in the plan and each operator do
      macro-gen := EBG(macro,op)
      world := post-conditions(macro-gen)
      if world = post-conditions(op) then
         if pre(macro) and pre(op) differ in one predicate
             ab := create-abstract-operator(macro,op)
             link(ab,op)
             link(ab,macro)
         operator-set := add-operator(ab, operator-set)
   return(operator-set)
```

Figure 2: Macro–Learner Algorithm

```
move-s($X $Y $Z)
   Pre: (ispeg $X)(ispeg $Y)(ispeg $Z)
        (ons $X)
  Post: (not ons $X)(ons $Z)

move-b($X $Y $Z)
   Pre: (ispeg $X)(ispeg $Y)(ispeg $Z)
        (ons $Y)
        (onb $X)
  Post: (not onb $X)(onb $Z)
```

AbTweak is used to find a solution for this problem which is the sequence **move-s(Peg1,Peg2), move-b(Peg1,Peg3), move-s(Peg2,Peg3)**. From this solution three macro sequences can be extracted.

```
seq-1: move-s(Peg1,Peg2),move-b(Peg1,Peg3)
seq-2: move-b(Peg1,Peg3),move-s(Peg2,Peg3)
seq-3: move-s(Peg1,Peg2),move-b(Peg1,Peg3),
       move-s(Peg2,Peg3)
```

From the first sequence **seq-1**, the following macro can be generated after using EBG to compute its pre–conditions and effects. Operator **macro1** is not changed when generalizing with the original operator **move-b** because neither restricts the variable instantiations.

```
macro1($V1 $V2 $V3)
     Pre:(is-peg $V1) (is-peg $V2)
         (is-peg $V3)
         (ons $V1) (onb $V1)
    Post:(ons $V2) (onb $V3)
         (not ons $V1) (not onb $V1)
```

The post–conditions of **macro1** and **move-b** are identical (except renaming of variables) and are given by the following set of facts:

```
(is-peg $V1) (is-peg $V2)(is-peg $V3)
(ons $V2)(onm $V3)
```

The algorithm then compares the pre–conditions of **macro1** and **move-b**. The pre–conditions differ only in the predicate **ons** which is **(ons $V1)** for **macro1** and **(ons $V2)** for **move-b**. Therefore, the macro–learner constructs an abstract operator in which the **ons** predicate is deleted. This abstract operator is generalized, and it contains two refinements: **move-b** and **macro1**. However, in order to avoid unnecessary variable instantiations, the linked macro is fully instantiated. In that way, if the same problem has to be solved in the future, the variables do not need to be re–instantiated. However, the abstract operator shows the generalization that is possible. Figure 3 shows the resulting operator hierarchy.

Since only **macro1** and **move-b** have identical post–conditions, the abstract operator in figure 3 is the only new operator that is added to the operator set.

Evaluation

With the implementation of the macro–learner, we tried to establish the usefulness of our first heuristic (to improve the operator set by generating new operators with more general pre–conditions). It is interesting to note that this heuristic leads to an abstraction hierarchy for the two–disk towers of Hanoi problem that is identical to the one shown to be optimal by Knoblock [Knoblock, 1991]. If the planner uses a control strategy that supports abstractions, the time complexity grows only linearly with the length of the solution [Knoblock, 1991].

Good performance of the planning system with the new operator set was expected, because the optimal set of abstractions was generated. This was verified in a number of experiments where the solution time was reduced from 40 to 24 seconds on a Sparc 2 station. It took ten seconds to compute the abstract operators. Similar results were obtained for the problem with the initial state **(ons peg3)(onb peg1)** and the goal state **(onb peg3)(ons peg3)**.

The macro–learner was also tested on the towers of Hanoi puzzle with three disks. The results of these experiments were similar to the ones for the previous example. The macro learner created two abstract operators:

- move the medium disk ignoring the small disk.

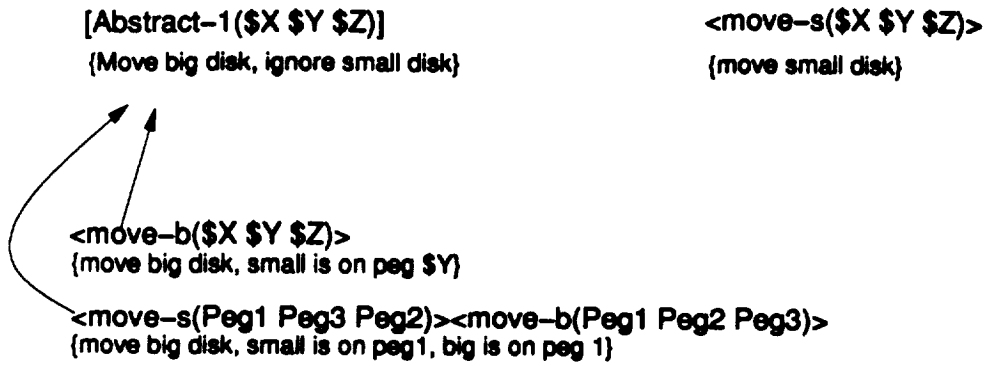

Figure 3: Learned Operator Hierarchy for the Towers of Hanoi 2

- move the large disk ignoring the medium disk.

These abstract operators are learned after only one successful plan is generated. After solving the problem a second time, the abstract operator learned to move the large disk regardless of where the medium and small disks are. These two abstract operators together with the primitive operator to move the small disk form the optimal operator set for the towers of hanoi problem with three disks. The solution time decreased from 2658 seconds to 29 seconds. The computation time for learning the abstract operators was 81 seconds. This result suggests that the learning time scales up much better than the planning time.

The most interesting result of the towers of Hanoi problem with two and three disks was that the system learned the optimal set of abstractions. This means that it not only learned the correct number of abstraction levels, but also the correct number of abstract operators for each level. Previous systems such as MPS, MAC-LEARN, and PiL2 are unable to learn these abstract operators.

Learning other Problem Solving Knowledge

This section describes methods for learning other problem solving strategies that can be represented in our common representation. One of the main advantages of a common representation is that not all operators in a refinement have the same level of abstraction. Therefore, other strategies such as reactive rules can be used. These strategies can be learned by comparing all refinements of a general operator.

After learning a new operator, the system uses additional heuristics to incorporate the new operator into the existing operator set. All new macros are part of the refinement of a more abstract operator (or the original initial state, goal state pair). The system compares the new operator to all other refinements of its abstract operator.

Raising Operators First the system tries to extract operators that occur in all refinements. The common operators are "raised" in the abstraction hierarchy, so that the planning system can focus on those operators. For example, given the abstract operator in figure 3, the operator to move the big disk is common in both refinements. In this case, the abstraction hierarchy is changed to reflect the fact that the operator `move-b` is an essential part of moving the big disk. The resulting abstract operator is similar to the RWM type operator subgoals [Güvenir and Ernst, 1990]. If the common operator occurs at the beginning of the sequence, a reactive rule is formed.

Generating Abstractions from Subsequences The system also extracts equivalent post-conditions resulting from the execution of operators in all refinements. If matching post-conditions are found, these states are extracted as new abstract operators. For example, assume that there are two operators to move the big disk, `move-b1` and `move-b2`. Furthermore, in figure 3, the system used `move-b1` in the first refinement and `move-b2` in the second refinement. In that case, there are no common operators in all refinements. Nevertheless, common to all refinements is a state where the small disk is on peg `$Y`. Therefore, the original problem is broken up into two abstract operators. The first one moves the small disk onto the medium peg, the second abstract operator moves the medium disk. The resulting operator hierarchy is identical to a subgoal sequence.

Failure When the system generates an unsuccessful plan, some of its expectations are wrong. If the user provides the system with additional information explaining why the plan failed (e.g. `(problem_x)`), the system can generate an abstract operator that relates the original problem to an elaboration of the problem, where the effects have additional conditions. (e.g. `(not problem_x)`). In the future, the planner is reminded of this problem and can avoid it.

Conclusions

The major contribution of this paper is the design of a learning system for a planner that combines macros, abstraction hierarchies, and case–based planning. The advantage of this approach is that techniques from both classical planning and case–based planning can be combined in the problem solving process.

The paper describes an analytical dynamic filtering scheme used to rule out inefficient operators. The dynamic filter is based on a formula relating the empirical *usefulness* and the length and branching factor of the operator. The common representation means that the dynamic filter can be applied to abstract operators and cases as well.

The paper also compares three previous approaches to the selection of new macros: Korf's MPS, Iba's MAC–LEARN, and Yamada's PiL2 system. From this comparison, guidelines are suggested for the selection of new operators. The goal in creating a new operator is trying to improve the current operator set. There are two ways in which an operator can be improved:

- Create an operator with more general pre–conditions. The effects of this operator can then be achieved in more states. The removal of predicates in pre–conditions leads to the generation of abstraction hierarchies.
- Create an operator with more specific effects. This removes side–effects of existing operators.

A macro learner was implemented and tested on a number of problems in the towers of Hanoi domain. As important parts of the complete planning systems are still missing, the implementation focused on comparing the learned macros to the ones learned by other systems. The results of the towers of Hanoi domain are promising.The system learned the optimal set of abstract operators for the two and three disk problems.

The paper also describes methods to learn diverse problem solving knowledge such as reactive rules, and automatic subgoaling. The next step in our research is the implementation of a complete planning system that incorporates these methods.

The paper also compares three different approaches to the selection of new macros. From this comparison guidelines are suggested for the selection of new operators. The main motivation for these heuristics is to find widely applicable operators with very specific effects.

References

Jacky Baltes. A symmetric version space algorithm for learning disjunctive string concepts. In *Proc. Fourth UNB Artificial Intelligence Symposium*, pages 55–65, Fredericton, New Brunswick, September 20–1 1991.

H. Altay Güvenir and George W. Ernst. Learning problem solving strategies using refinement and macro generation. *Artificial Intelligence*, 44(3):209–243, 1990.

Kristian J. Hammond. *Case Based Planning.* Academic Press Inc., 1989.

G. A. Iba. A heuristic approach to the discovery of macro–operators. *Machine Learning*, 3(4):285–318, 1989.

Craig A. Knoblock. *Automatically Generating Abstractions for Problem Solving.* PhD thesis, School of Computer Science, Carnegie Mellon University, 1991. Tech. Report CMU-CS-91-120.

R. E. Korf. Macro–operators: A weak method for learning. *Artificial Intelligence*, 26(1):35–77, 1985.

R. E. Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33(1):65–88, 1987.

S. Minton. Selectively generalizing plans for problem solving. In *Proceedings of the Ninth International Conference on Artificial Intelligence*, pages 595–599, 1985.

P. Shell and J. Carbonnel. Towards a general framework for composing disjunctive and interative macro–operators. In N. S. Sridharan, editor, *IJCAI–89 Proceedings of the Eleventh International Joint Conference on Articifical Intelligence*, volume 1, pages 596–602, 1989.

S. Yamada and S. Tsuji. Selective learning of macro–operators with perfect causality. In N. S. Sridharan, editor, *IJCAI–89 Proceedings of the Eleventh International Joint Conference on Articifical Intelligence*, volume 1, pages 603–608, 1989.

Qiang Yang and Josh D. Tenenberg. ABTWEAK: Abstracting a nonlinear, least commitment planner. Technical report, University of Waterloo, 1990.