Constraint Limited Generalization: Acquiring Procedures From Examples

Peter M. Andreae

M.I.T. Artificial Intelligence Laboratory
545 Technology Sq., Cambridge, MA 02139

Abstract

Generalization is an essential part of any system that can acquire knowledge from examples. I argue that generalization must be limited by a variety of constraints in order to be useful. This paper gives three principles on how generalization processes should be constrained. It also describes a system for acquiring procedures from examples which is based on these principles and is used to illustrate them.

1. Introduction.

Much of the work on learning in AI can be viewed as an attempt to understand the problem of generalization in a variety of domains. Much of it has been concept learning—acquiring descriptions of some concept from descriptions of particular examples of the concept. A set of standard heuristics for concept acquisition which are applicable in a wide range of domains are given in Winston [1970] and Michalski [1983].

Finding a generalization of a set of examples is an unconstrained problem—there are usually many descriptions which are valid generalizations of the examples. In any interesting domain, the number of possible generalizations is too great to consider them all. Many generalization systems have constrained the search space by ordering it according to generality, and choosing the most specific generalization that is consistent with the examples. For many interesting domains, this constraint is not sufficient to make generalization tractable, and further constraints on the generalization process must be found.

This paper provides three principles to help determine these constraints:

- **Domain Constrained Generalization:** All possible constraints from the domain should be used to eliminate possible generalizations and reduce the search space.
- **Undesirability Ordering:** There must be some ordering on the space of possible descriptions that represents their relative desirability, based not only upon generality, but also upon the shape of the search space and the expressive power of the representation language.
- **Context Limited Generalization:** Given an undesirability ordering, the context of two items being generalized must constrain the set of generalizations that should be considered. Consideration of more undesirable generalizations requires stronger justification from the context.

These principles are best discussed in the context of a particular generalization task. Therefore section 2 will describe a system called PMA[**], for acquiring procedures by generalizing from examples, which is based on these principles. Section 3 will discuss the principles in more detail using examples from the system.

2. Acquiring Procedures from Examples.

In the standard concept acquisition task, a teacher provides the learner with a series of examples (and possibly non-examples) of a concept. The learner must generalize these examples to obtain a description of the concept from which the examples were derived.

The procedure acquisition task is similar: a teacher provides the learner with a series of traces of the execution of a procedure. Each trace will show the operation of the procedure in one particular set of circumstances. The learner must generalize the traces to obtain a description of the procedure that will apply under all circumstances—the procedure that the teacher was using to generate the traces.

For example, the teacher may show a robot how to assemble a device in several different cases: perhaps the normal case, the case when the parts are not found in the usual position, the case when the washer sticks during assembly, and the case when the screw holes are not aligned correctly. For each case, the teacher will lead the robot through the entire assembly task, and each trace will consist of the sequence of actions and the feedback patterns after each action. From this, the robot should acquire the complete assembly procedure.

Several people (*e.g.*, Mitchell [1983], Langley [1983] and Anderson [1983]) have approached related problems using a production system representation of the procedure being acquired. Here, we wish to acquire procedures with explicit control structure. This control structure—sequencing, branching, loops, and variable reference—is not present in the example traces and must be inferred. Therefore, we cannot use the generalization methods used in in either concept or production system acquisition in a straightforward manner.

Acquisition of procedures with explicit control structure has also been studied by Van Lehn [1983] (a multi-column subtraction procedure) and Latombe [1983] (a robot "peg-in-hole" procedure). The induction of finite state automata from regular strings, and the induction of functions from input/output pairs (see Angluin and Smith [1982]) is related to procedure acquisition, but the goals and methods in these tasks are sufficiently different that they will not be discussed here.

2.1 Domain.

PMA embodies a procedure acquisition algorithm that is intended to apply to a wide variety of domains. For each domain, there is a set of legal *actions* that can be performed in the domain and the feedback *patterns* that will result from the actions. PMA is designed to acquire procedures in any domain in which the actions are specified by an action type and a set of parameters, (not necessarily numerical), and the patterns consist of a set of pattern components, each component being specified by a pattern type and a list of parameters.

All the examples given below will be taken from a simple two dimensional robot world which meets these criteria. The primitive actions of this robot domain include `MOVE`, `MOVE-UNTIL-CONTACT`, `ROTATE`, `GRASP` and `UNGRASP`. The parameters of the `MOVE` and `MOVE-UNTIL-CONTACT` actions consist of a vector specifying the distance and direction of the move, the parameter of the `ROTATE` action is an angle, and the other two actions have an empty parameter list.

The *pattern* that the robot world returns in response to an action has three components: the new `POSITION` of the robot, given in x-y coordinates; its `ORIENTATION`, specified by an angle; and the `CONTACT`, if any, between the robot and an obstacle, specified by direction of the obstacle.

2.2 Representation.

The traces (or examples) given to PMA by the teacher are sequences of alternating *actions* and *patterns* starting with a `START` action and ending with a `STOP` action. They will be represented by a sequence of *events*, each event containing a pattern and the following action. Figure 1 shows two traces the teacher might provide to teach a simple turtle procedure for circumnavigating obstacles. The turtle procedure is: "**Move towards goal; if you hit something, move perpendicularly away from the obstacle $\frac{1}{2}$ step, move to the side 1 step, and try again.**" The first trace results from the application of the procedure when no obstacles are present, and the second, when one small obstacle is present.

[*]This paper reports work done at the Artificial Intelligence Laboratory of the Massachusetts Institute Of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N0014-80-C-0505

[**]Procedure Matcher and Acquirer.

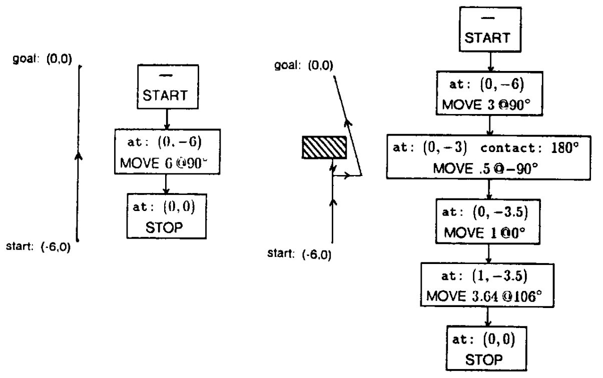

Figure 1. Two Turtle Traces.

The procedures that PMA must infer from the traces may have conditional branching, iteration (loops), variables, and generalized actions that may specify a class of primitive actions. We will represent these procedures by a directed graph rather like the usual graphical representation of a finite state automata. Each node of the graph is marked with an event which specifies the *condition* under which control can pass to the event from the previous event, and an *action* to perform if control does reach the event. The conditions are generalizations of the patterns in the traces. The condition may also assign variables to parameters of the pattern for use in later actions. One event, which has no links into it, is distinguished as the *start event* and contains a null condition and a START action. Conditional branching is represented by multiple edges or *links* proceeding from one event, and iteration or looping is represented as a cycle in in the graph. Generalized actions are represented in the same way as the primitive actions, *i.e.*, by an action type and associated parameters.

Figure 2 shows the representation for the turtle procedure. The MOVE-UNTIL-CONTACT-TOWARD (0,0) action is an example of a generalized action—it is not one of the primitive actions of the domain. It specifies whatever MOVE-UNTIL-CONTACT action will move from the current position toward the position (0,0). The event containing this action is followed by a conditional branch: if the position after the MOVE-UNTIL-CONTACT-TOWARD action is [at: (0,0)] then the left branch will be taken and the START action performed. If the position is anywhere other than (0,0), and there is a contact at some angle [contact: ⟨any-angle⟩], then the actual angle of contact will be stored in the variable θ and the right branch will be taken, entering the loop. If neither condition is met, the procedure fails. The directions of two MOVE actions in the loop are specified in terms of functions of the angle θ.

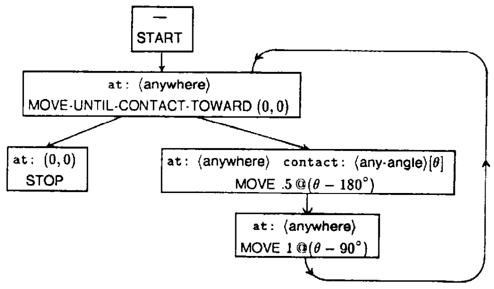

Figure 2. Turtle Procedure

PMA can infer procedures like that of figure 2 from traces like those of figure 1. The following sections outline the matching and generalization methods that it uses.

2.3 Matching and Generalizing.

PMA operates incrementally on two levels. Like Winston's [1970] concept learner, PMA builds its description of the goal procedure incrementally, taking one new trace at a time and generalizing its current description of the procedure to incorporate the new trace. Its initial description of the procedure will just be the first trace.

PMA also processes each new trace incrementally. To incorporate a new trace, PMA matches the current procedure and the new trace to find a pairing between the procedure events and the trace events. It notes any differences and generalizes the procedure to eliminate the differences. However, the matching and generalizing is done in several stages. The initial stage of matching the procedure and the trace does no generalization of the individual events in the procedure and finds only a *skeleton match* that pairs procedure and trace events that match exactly. This provides the context for the second matching stage that does generalize procedure events, if necessary, to find a more complete pairing of procedure and trace events. This, in turn, provides the context for further stages which can perform more powerful generalizations in the appropriate circumstances. This incremental generalization is based on the principle of **context limited generalization**—the more powerful generalization methods are only applied in the context of the match produced by the less powerful methods. Since the later stages depend upon the correctness of the earlier stages, it is important that the earlier stages do not find any spurious pairings of events. Therefore, PMA must only attempt to match two events when there is good justification for doing so.

To avoid spurious pairings in the skeleton match, PMA only searches for pairings involving the events of the procedure for which reliable matches can be found—the *key events*. The START and STOP events are obvious candidates for the *key events*. Figure 3 shows the skeleton match of the traces of figure 1 using these key events. In more complex procedures, the key events may also include *unique events* (events of which the action type occurs only once in the procedure), and *bottleneck events* (sequences of events at the merging of several branches through which control always flows).

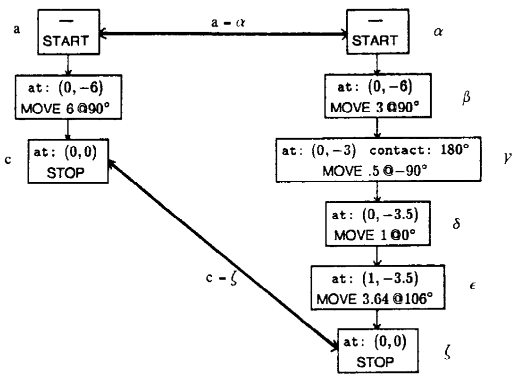

Figure 3. Skeleton Match

2.4 Second Stage—Propagation and Event Generalization.

The second stage builds on the skeleton match by pairing procedure and trace events found by propagating through the procedure and trace starting from the pairs found in the skeleton match. The propagation exploits the sequential structure of procedures in order to find justifiable pairs in much the same way as Winston's analogy program [Winston 1984] exploits the causal structure of stories. Figure 4 illustrates this, building on the skeleton match of figure 3. The pair a-α was found in the skeleton match. Since b and β were the respective successors of a and α, PMA paired b and β. Propagating from b-β, PMA attempted to pair c and γ. PMA also propagated *backwards* from c-ζ to find the pair b-ϵ, and attempted to pair a and δ.

For this stage, if the procedure and trace events being paired only match partially, PMA attempts to find a generalization of the two events to place in the new procedure. If no generalization can be found, then the pair is abandoned. In figure 4, b and β did not match exactly, but PMA found a generalization of them, as shown at the bottom of the figure. When it attempted to pair c and γ, not only were they not equal, but there was no possible generalization of the two events, so the pairing was abandoned. Similarly, in propagating backwards, PMA found a generalization of b and ϵ but not for a and δ.

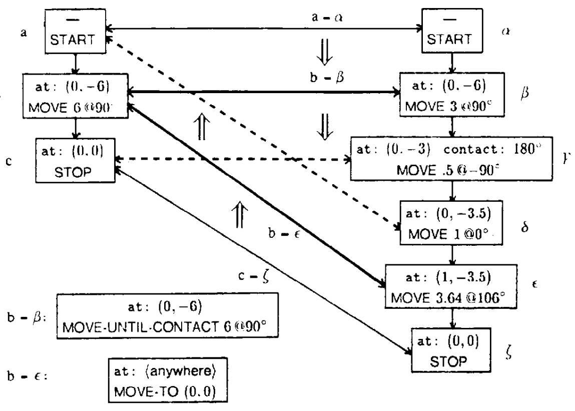

Figure 4. Propagation and Event Generalization

The propagation stage is completed by several bookkeeping steps. Pairs that involve the same events are grouped and generalized, and events from the procedure and the trace that have not been paired are collected. In the example of figure 4, the event b is involved in two pairs, (b-β and b-ϵ), which are then matched and generalized. The events γ and δ were not paired with any other events, so they are simply installed into the new procedure along with b-β-ϵ and the skeleton pairs a-α and c-ζ. The new procedure is shown in figure 5.

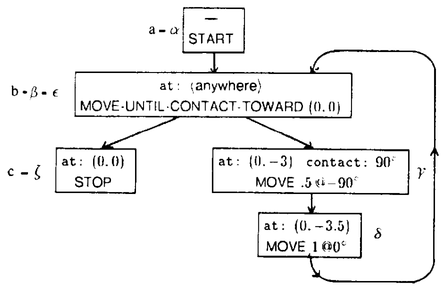

Figure 5. New Procedure

The matching and generalizing of events is done with reference to the action and condition hierarchies. These hierarchies are partially ordered graphs where each node describes a generalized action or condition. Each action hierarchy corresponds to one of the classes of action specified by the domain, and the base of the hierarchy is the primitive action of that class. Similarly, each condition hierarchy corresponds to one component of the pattern specified by the domain, and the base of a condition hierarchy is a primitive pattern that occurs in the traces. Every higher node of a hierarchy is a generalized action type or condition. Figure 6 shows part of the MOVE action hierarchy. Each node describes the type of the action or condition and the parameters associated with it. For example, the MOVE-TO node in figure 6 has a position parameter (x, y). Also attached to each node, but not shown in figure 6, are procedures for determining whether an instance of the node is a generalization of an instance of a lower node and constructor procedures for creating generalizations of two instances of lower nodes.

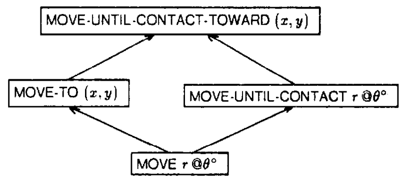

Figure 6. MOVE Action Hierarchy

If PMA is given two actions to match, it will first determine if they are of the same class. If not, the match immediately fails. Otherwise, it will determine their types, and the relative position of the nodes of those types in the appropriate action hierarchy. If the actions are both instances of the same node (*i.e.*, they are of the same type), it simply tests equality of the parameters. If one node is a direct superior of the other, it will invoke the appropriate procedure attached to the higher node to determine whether the first action is a direct generalization of the second. If either of these tests fail, or neither node is a direct superior of the other, it will search for a node that is a direct superior of both nodes and invoke the constructor procedure to create a generalization of the two actions. Often the constructor procedure will not return a generalization, in which case the match fails. The same process is followed in matching patterns and/or conditions.

Each new domain in which PMA is to be used will require a different set of action and condition hierarchies, since they are obviously domain dependent. However, the structure of the hierarchies and the way they are used in matching and generalizing events remains the same across domains. Furthermore, the hierarchies themselves could be acquired from the traces provided by the teacher. How this might be done will be discussed briefly in a later section. The same mechanism could also be used to extend hierarchies which have been previously specified to incorporate new generalized actions or conditions that are needed for particular classes of procedures.

2.5 Third Stage—Function Induction.

The third stage of matching and generalizing searches for a particular configuration—*parallel segments*—in the description of the procedure produced by the first two stages. There are no parallel segments in the procedure of figure 5, but there are in the procedure of figure 8 which was generated by applying the first two stages to the procedure of figure 5 and the new trace shown in figure 7. A *segment* is a sequence of connected events with no branching. Two segments of a procedure are *parallel* if they start and end at the same events, they contain the same number of events, the corresponding conditions match, and the corresponding actions are of the same type. Parallel segments represent events that the second stage attempted to pair but abandoned because it could find no generalizations of the actions using the action

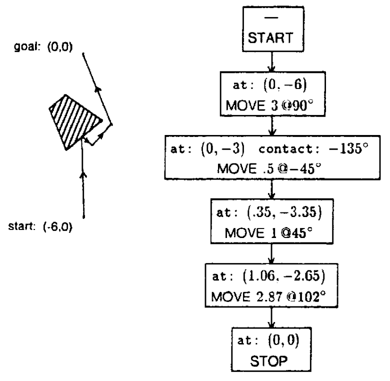

Figure 7. Third Turtle Trace

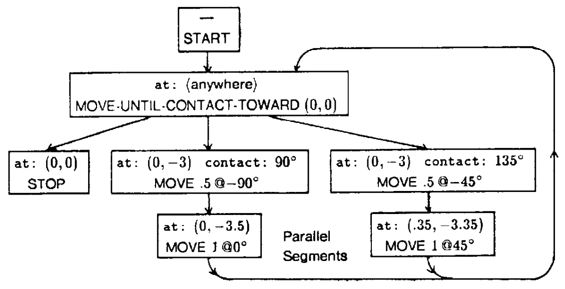

Figure 8. Procedure with Parallel Segments

hierarchy. The corresponding MOVE actions in the parallel segments of figure 8 cannot be merged without reference to the contact angle in an earlier pattern upon which the directions of the MOVE's depend. No generalization in the action hierarchy could express this dependency.

The identical context of the parallel segments suggests that they play the same role in the procedure. With this justification, the third stage applies a more powerful generalization method which attempts to match the events by searching for functional dependencies of actions upon earlier patterns. In the example of figure 8, the two pairs of **MOVE** actions should be generalized to MOVE's whose direction is given by the earlier contact angle minus $180°$ and $90°$ respectively, as shown in the procedure of figure 2. These functional expressions are simple and are found readily. When the parallel segments are merged by this third stage, the resulting procedure is exactly the goal procedure of figure 2.

Finding the functional dependencies involves a double search to find both an earlier pattern component on which the actions may depend and also the function relating the pattern component to the action. To avoid finding spurious functional relationships, PMA searches for the condition closest to the actions for which it can find a functional relation.

For each candidate condition component that the first search considers, PMA searches the space of possible functions that fit the past values of the condition components and the corresponding values of the actions being merged. (Note that this requires that PMA retain a certain amount of information about the past values of the patterns and actions from which the generalized conditions and actions were constructed). The space of functions is searched by incrementally building expressions from a known set of operators. The choice of operators is constrained by the type of the input and output values (positions, angles, numbers, lists, *etc.*), which requires that the types of the arguments and ranges of every operator must be known.

The algorithm initially considers expressions containing a single operator applied to the domain values (from the condition) which returns the range values (from the action). If none are found, it will recursively apply any appropriate operators to the domain and range values, and search for an "connecting" operator which returns the new range values when applied to the new domain values. The resulting expression will be the composition of the inverses of the operators applied to the range values, the connecting operator, and the operators applied to the domain values. The search fails when it cannot find an expression within some complexity limit.

Functions involving constants pose a problem for function induction, since it is not possible to search the space of all possible values of constants if the space is infinite, as in a domain involving real-valued parameters such as the robot domain. PMA's algorithm solves this problem by only considering one new constant for each expression. Such a constant can be found if applying an operator to each pair of the domain and range values produces a constant value. For example, when the difference operator is applied to the pairs of angles $(90°, -90°)$ and $(135°, -45°)$, the result is $180°$ for both pairs. The required expression can be found by inverting the difference operator and using the constant $180°$ to obtain the expression: *move-direction* = (- *contact-angle* $180°$). If there are any constants with predetermined values which may be relevant to the functional dependency, these can also be included in the candidate expressions. One source for such *known constants* is the condition immediately preceding the actions being merged. The possible relevance of the parameters of this condition is justified by the fact that this condition represents information about the state of the world in which the action is to be performed.

The algorithm for searching through the space of functions relies only on being provided with a set of invertible operators whose domain and range types are specified. The operators need not be numeric (although they are for the robot domain) and the algorithm is therefore quite domain independent.

The generalization of the third stage is more powerful than that of the second stage, both because it involves two events simultaneously and because the space of possible functions is very large. In fact, if the space is unconstrained, it will be possible to find a functional relation between almost any pattern and action. For this reason, the functional generalization is only applied in the context of parallel segments and the complexity of the expressions that are considered must be constrained by the number of data points available.

At this point we note that some of the generalized actions (*e.g.*, MOVE-TO (x, y)) are actually primitive actions whose parameters are a function of the immediately preceding pattern. These nodes in the action hierarchy are essentially memoized forms of these "local" functional relationships. The action hierarchy can be augmented by noting reoccurring actions with the same local functions and constructing the memoized form of the function. This has not yet been implemented in PMA.

2.6 Final Stage—Consistency checking.

The final stage of PMA checks that the description produced by the first three stages satisfies the constraint that valid procedures must be deterministic, *i.e.*, at every step, the procedure must specify exactly one action. This constraint may be violated if the conditions at a conditional branch are not sufficiently distinct. If there are no possible patterns that would match more than one of the branching conditions, then the branch satisfies the constraint. If there is a pattern which matches two conditions, then the branch may be indeterminate. We adopt the convention that if one of the conditions is a strict generalization of the other, then control passes to the most specific. This convention eliminates the need for conditions with complex exception clauses. If this is not the case—either the two conditions are are identical or part of one condition is a generalization and part is a specialization of the other condition—then the branch violates the constraint, and must be rectified.

There are several ways a non-deterministic branch could arise, each representing a different way of resolving the non-determinism. One source is that the second stage was not able to find a generalization of the two actions of the events involving the conflicting conditions. PMA therefore attempts to generalize the actions by searching for a functional dependency as in the third stage. If this is successful, the events can be combined, and the indeterminacy removed. If this is not successful, it will attempt to specialize the conditions on the assumption that the second stage may have over-generalized them. This is done by searching in the condition hierarchy for a node lower than the current condition. For example, it might be that some action should be performed only when the position is within some circular region. If the initial traces contain the action occurring in several positions, PMA will generalize the condition to [at: ⟨anywhere⟩]. When later traces show a different action occurring at other positions, PMA will have to specialize the original condition to [inside: ⟨circle-1⟩].

If no specialization node is found in the condition hierarchy, it may be possible to create a new node using standard concept acquisition techniques. For example, if there were no circle node, one could be created, using the positions associated with the first action as the positive examples of the new concept and the positions associated with the other action as the near misses. In a domain like the robot world involving numerical and geometric parameters, it may be possible to use an algorithm similar to the function induction algorithm to create the expressions representing the new concepts. This has not been implemented in PMA.

If neither of these methods eliminates the indeterminate branch, the pairing that created the branch event will be "undone". This may have to be repeated until all indeterminate branches are removed.

3. Discussion.

The need for constraints on generalization is not a new idea. Winston [1970] constrained his concept learner to always choose the most specific generalization consistent with the examples. Furthermore, it was constrained to ignore negative examples unless there was exactly one difference from the current concept, indicating an unambiguous change to the current concept. These constraints reduced the search by avoiding the need for backtracking, which is very expensive. Mitchell's [1982] version space algorithm relaxed these constraints by providing an efficient characterization of an entire set of generalizations consistent with the examples. This fails, however, if disjunctions are allowed in the descriptions, and further constraints are necessary.

Efficiency is not the only reason for constraints. In some cases, the generalization task is so under-specified that additional constraints must be found in order to perform the task at all. A good example is Berwick's language learner [1982] which acquires grammar rules when given only grammatical sentences and no negative examples. It was only by adopting a particular parser and the very strict constraints on the form of its rules that it was possible to learn any grammar rules strictly from positive examples.

The three principles stated in the introduction describe three classes of constraints on generalization which apply to any generalization task.

3.1 Domain Constrained Generalization.

Exploiting the constraints of the domain is an important and established technique for all areas of AI. Domain constraints may reduce the search space by eliminating descriptions that can validly be generated by the representation language, but describe situations that are illegal in the domain. PMA exploits the constraint that procedures must be deterministic to eliminate any descriptions with non-deterministic branches. This determinacy constraint also reduces the space of legal generalized actions. Although the action [MOVE-TO (0,0)] represents many possible primitive MOVEs, in any particular situation (*i.e.*, from any particular position) it specifies exactly one. However, the action [MOVE 1@(any-angle)] is indeterminate in that it never specifies a particular primitive action and the determinacy constraint therefore eliminates it from consideration.

Domain constraints may also be used to guide the generalization process in ways other than simply reducing the search space. For example, it is a particular property of the robot domain that MOVEs and MOVE-UNTIL-CONTACTSs are very closely related, though they are actually different primitive actions. PMA exploits this relation by treating MOVE-UNTIL-CONTACT as a generalization of MOVE, and is able to determine when a particular MOVE made by the teacher was intended to be a MOVE-UNTIL-CONTACT. This type of generalization is very domain specific, but illustrates the way in which particular properties of a domain can be used to increase the power of the generalizer.

3.2 Undesirability Ordering.

In order to guide the generalization processes, some ordering must be placed on the space of possible descriptions. Generally, out of a set of descriptions that are all valid generalizations of a set of examples, one chooses the description that is lowest in the ordering. This is particularly important for acquisition tasks in which no negative examples are given. In most concept acquisition programs, this ordering has been based on either generality or complexity—the more general (or complex) the description, the more undesirable it is.

This is sufficient for restricted domains, such as those in which all the concepts that need to be considered can be described in terms of a conjunctive list of properties of, and relations between objects. In domains involving descriptions based on a more powerful description language, however, this undesirability ordering must involve more than just generality or complexity. For example, if the description language allows disjunction, there will always be a generalization of any two examples consisting of the disjunction of the descriptions of the two examples. This is the most specific generalization possible, but it is seldom a useful or desirable generalization. Similarly, there is always an $(n-1)$ degree polynomial that fits n points on the plane, but it is seldom a useful generalization of the points. Neither of these generalizations is useful because the existence of such a generalization was a foregone conclusion, whether the relation between the examples was significant or entirely random. If, however, there were a conjunctive description, or a low degree polynomial, this would describe a relation between the examples which would not be true of a random set of examples. Both disjunctive descriptions and high order polynomials are necessary at times, and cannot be eliminated from the search space entirely, but should be placed high on the undesirability ordering.

The common element of these two undesirable generalizations is that they use representation constructs that are very "powerful" in the sense that they allow one to construct descriptions of any set of items, whereas conjunctive descriptions or 2nd degree polynomials can only describe some sets of items. In other words, the space of possible descriptions is very much wider if disjunctions, or other powerful constructs are allowed than if they are prohibited. The undesirability ordering must therefore take into account the descriptive power of the components of the representation language, and place descriptions using the more powerful constructs higher than those with more restrictive constructs.

PMA must be able to acquire procedures that involve conditional branching, which is a form of disjunction. However, following this principle of undesirability ordering, it always chooses a procedure without branches over one with branches even at the cost of more general events or actions containing functional expressions. Similarly, although an explicit functional expression in an action is not any more more general than an action from the action hierarchy, PMA always prefers an action from the hierarchy, if one exists, because the description language for actions in the action hierarchies is less powerful than that for arbitrary functions, and therefore lower in the undesirability ordering.

3.3 Context Limited Generalization.

However, it is not sufficient to simply order the generalizations by undesirability and choose the least undesirable. Matching two descriptions involves finding a pairing between the elements of the descriptions. With a sufficiently powerful description language, a generalization can be found for any pair of the elements. But most of these pairings will be spurious. We need to place some limit on the degree of undesirability to which we are prepared to go, but this limit must not eliminate undesirable generalizations that really are part of the match. The solution is to use a limit that varies with the justification for believing that a generalization exists. If a teacher has asserted that two situations match, then there is good justification for resorting to a very undesirable generalization of the two situations. On the other hand, when searching a data base for possibly relevant situations to a problem at hand, then only generalizations low in the order should be considered. When matching two structures, the partially completed match consisting of pairings of very similar components may provide a context that justifies considering a very undesirable generalization of two components that fill corresponding positions according to the pairings found so far. In general, the generalization must be limited to a level of undesirability consistent with the context in which the generalization takes place.

The several stages of PMA illustrate this principle well. In the first stage, there is no context to suggest what pairings should be made, and therefore no event generalization at all is allowed. In the second stage, the context of the perfectly matched pairs gives more justification to the pairs found by propagation, so generalized events are considered. The function induction is only considered in the highly restricted contexts of parallel segments or indeterminate branches. If function induction were allowed in the second stage it would be very likely to find spurious generalizations. But with this context limited generalization, PMA is able to use powerful generalization methods without producing spurious matches.

References.

Anderson, J.R. [1983]; *Acquisition of Proof Skills in Geometry*; in "Machine Learning", *eds.* Michalski, Carbonell, Mitchell, Tioga Pub Co., Palo Alto, California.

Angluin, D. and C.H. Smith [1982]; *A Brief Survey of Inductive Inference*; Technical Report 250, Dept. Comp. Sci., Yale University.

Berwick, R.R. [1982]; *Locality Principles and the Acquisition of Syntactic Knowledge*; PhD thesis, M.I.T..

Langley, P, [1983]; *Learning Effective Search Heuristics*; Proceedings of IJCAI-83, Vol 1, 419-421.

Latombe, J-C. and Dufay, B. [1983]; *An Approach to Automatic Robot Programming Based on Inductive Learning*; Robotics Workshop, M.I.T..

Mitchell, T.M. [1982]; *Generalization as Search*; Artificial Intelligence, Vol. 18, 203-226.

Mitchell, T.M. [1983]; *Learning and Problem Solving*; Proceedings of IJCAI-83, Vol 2, 1139-1151.

Michalski, R.S. [1983]; *A Theory and Methodology of Inductive Learning*; in "Machine Learning" *eds.* Michalski, Carbonell, Mitchell, Tioga Pub. Co., Palo Alto, California.

Winston, P.H. [1970]; *Learning Structural Descriptions From Examples*; PhD Thesis, M.I.T..

Winston, P.H. [1984]; *Artificial Intelligence*; Ch 12, Addison Wesley, Reading, Massachusetts.

Van Lehn, Kurt [1983]; *Felicity Condition for Human Skill Acquisition: Validating an AI-based Theory*; PhD Thesis, M.I.T..