December 1987 UILU-ENG-87-2277

COORDINATED SCIENCE LABORATORY
College of Engineering

ACQUIRING GENERAL ITERATIVE CONCEPTS BY REFORMULATING EXPLANATIONS OF OBSERVED EXAMPLES

Jude W. Shavlik
Gerald F. DeJong

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-87-2277	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois 6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION Office of Naval Research National Science Foundation
6c. ADDRESS (City, State, and ZIP Code) 1101 W. Springfield Avenue Urbana, IL 61801	7b. ADDRESS (City, State, and ZIP Code) 800 N. Quincy St., Arlington, VA 22217 1800 G. Street N.W., Washington, D.C. 20550
8a. NAME OF FUNDING/SPONSORING ORGANIZATION ONR/NSF 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-86-K-0309 NSF IST 85-11542
8c. ADDRESS (City, State, and ZIP Code) 800 N. Quincy St., Arlington, VA 22217 1800 G. Street N.W., Washington, D.C. 20550	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Acquiring General Iterative Concepts by Reformulating Explanations of Observed Examples

12. PERSONAL AUTHOR(S)
Shavlik, Jude W., DeJong, Gerald F.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM _____ TO _____	December 1987	54

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	artificial intelligence, machine learning, explanation-based learning, generalizing number, generalizing to N, BAGGER, empirical analysis

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Most research in *explanation-based learning* involves relaxing constraints on the variables in the explanation of a specific example, rather than generalizing the *structure* of the explanation itself. However, this precludes the acquisition of concepts where an iterative process is implicitly represented in the explanation by a fixed number of applications. Such explanations must be reformulated during generalization. The fully-implemented **BAGGER** system analyzes explanation structures and detects extendible repeated, inter-dependent applications of rules. When any are found, the explanation is extended so that an arbitrary number of repeated applications of the original rule are supported. The final structure is then generalized and a new rule produced which embodies a crucial shift in representation. An important property of the extended rules is that their preconditions are expressed in terms of the initial state — they do not depend on the results of intermediate applications of the original rule. **BAGGER**'s generalization algorithm is presented and empirical results that demonstrate the value of generalizing to N are reported. To illustrate the approach, the acquisition of a plan for building towers of arbitrary height is discussed in detail.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

Technical Report
UILU-ENG-87-2277

Acquiring General Iterative Concepts by Reformulating Explanations of Observed Examples[†]

Jude W. Shavlik[‡]
Gerald F. DeJong

Coordinated Science Laboratory
University of Illinois
Urbana, IL 61801 USA

Abstract

This is an extended version of a chapter appearing in *Machine Learning: An Artifical Intelligence Approach*, Volume III, R. Michalski and Y. Kodratoff (eds.), Morgan-Kauffman, 1988.

Most research in *explanation-based learning* involves relaxing constraints on the variables in the explanation of a specific example, rather than generalizing the *structure* of the explanation itself. However, this precludes the acquisition of concepts where an iterative process is implicitly represented in the explanation by a fixed number of applications. Such explanations must be reformulated during generalization. The fully-implemented **BAGGER** system analyzes explanation structures and detects extendible repeated, inter-dependent applications of rules. When any are found, the explanation is extended so that an arbitrary number of repeated applications of the original rule are supported. The final structure is then generalized and a new rule produced which embodies a crucial shift in representation. An important property of the extended rules is that their preconditions are expressed in terms of the initial state — they do not depend on the results of intermediate applications of the original rule. **BAGGER**'s generalization algorithm is presented and empirical results that demonstrate the value of generalizing to N are reported. To illustrate the approach, the acquisition of a plan for building towers of arbitrary height is discussed in detail.

[†] This research was partially supported by the Office of Naval Research under grant N00014-86-K-0309, by the National Science Foundation under grant NSF IST 85-11542, and by a University of Illinois Cognitive Science/Artificial Intelligence Fellowship to the first author.

[‡] Current address: Computer Sciences Department, University of Wisconsin, Madison, WI, 53706, USA.

1. INTRODUCTION

Often an expert will, in the course of solving a problem, repeatedly employ an action or collection of actions. It is an important, but difficult, problem, to correctly generalize this sequence once observed. Sometimes the number of repetitions itself should be the subject of generalization. Other times it is quite inappropriate to alter the number of repetitions. This article addresses the important issue in explanation-based learning (EBL) of *generalizing to N* (Shavlik and DeJong, 1985, 1987b, 1987c). This can involve generalizing such things as the number of entities involved in a concept or the number of times some action is performed. Generalizing number has been largely ignored in previous explanation-based learning research. Instead, other research has focused on changing constants into variables and determining the general constraints on those variables.

In explanation-based learning (DeJong and Mooney, 1986; Ellman, 1987; Mitchell, Keller, and Kedar-Cabelli, 1986) a specific problem solution is generalized into a form that can be later used to solve conceptually similar problems. The generalization process is driven by the *explanation* of why the solution worked. Knowledge about the domain allows the explanation to be developed and then generalized.

Consider the **LEAP** system (Mitchell, Mahadevan, and Steinberg, 1985). The system is shown an example of using *NOR* gates to compute the boolean *AND* of two *OR*'s. It discovers that the technique generalizes to computing the boolean *AND* of any two inverted boolean functions. However, **LEAP** cannot generalize this technique to allow constructing the *AND* of an arbitrary number of inverted boolean functions using a multi-input *NOR* gate. This is the case even if **LEAP**'s initial background knowledge were to include the general version of DeMorgan's Law and the concept of multi-input *NOR* gates. Generalizing the number of functions requires alteration of the original example's explanation.

Ellman's (1985) system also illustrates the need for generalizing number. From an example of a four-bit circular shift register, his system constructs a generalized design for an arbitrary four-bit permutation register. A design for an *N*-bit circular shift register cannot be produced. As Ellman points out, such generalization, though desirable, cannot be done using the technique of changing constants to variables.

Many important concepts, in order to be properly learned, require generalization of number. For example, physical laws such as momentum and energy conservation apply to arbitrary numbers of objects, constructing towers of blocks requires an arbitrary number of repeated

stacking actions, and setting a table involves a range of possible numbers of guests. In addition, there is recent psychological evidence (Ahn, Mooney, Brewer, and DeJong, 1987) that people can generalize number on the basis of one example.

Repetition of an action is not a sufficient condition for generalization to N to be appropriate. Compare two simple examples. Generalizing to N is necessary in one but inappropriate in the other. The examples are:

- observing a previously unknown method of moving an obstructed block, and
- seeing, for the first time, a toy wagon being built.

Suppose a learning system observes an expert achieving the desired states. In each case, consider what general concept should be acquired.

In the first example, the expert wishes to move, using a robot manipulator, a block which has four other blocks stacked in a tower on top of it. The manipulator can pick up only one block at a time. The expert's solution is to move all four of the blocks in turn to some other location. After the underlying block has been cleared, it is moved. In the second example, the expert wishes to construct a movable rectangular platform, one that is stable while supporting any load whose center of mass is over the platform. Given the platform and a bin containing two axles and four wheels, the expert's solution is to first attach each of the axles to the platform. Next all four of the wheels are grabbed in turn and mounted on an axle protrusion.

This comparison illustrates an important problem in explanation-based learning. Generalizing the block unstacking example should produce a plan for unstacking *any* number of obstructing blocks, not just four as observed. The wagon-building example, however, should not generalize the number "4." It makes no difference whether the system is given a bin of five, six, or 100 wheels, because only four wheels are needed to fulfill the functional requirements of a stable wagon.

Standard explanation-based learning algorithms (DeJong and Mooney, 1986; Fikes, Hart, and Nilsson, 1972; Hirsh, 1987; Kedar-Cabelli and McCarty, 1987; Mitchell, Keller, and Kedar-Cabelli, 1986; Mooney and Bennett, 1986; O'Rorke, 1987a) and similar algorithms for chunking (Laird, Rosenbloom, and Newell, 1986) cannot treat these cases differently. These algorithms, possibly after pruning the explanation to eliminate irrelevant parts, replace constants with constrained variables. They cannot significantly augment the explanation during generalization. Thus, the *building-a-wagon* type of concept will be correctly acquired but the *unstacking-to-move* concept will

be undergeneralized. The acquired schema will have generalized the identity of the blocks so that the target block need not be occluded by the same four blocks as in the example. Any four obstructing blocks can be unstacked. However, there must be exactly four blocks.[1] Unstacking five or more blocks is beyond the scope of the acquired concept.

Note that EBL systems do not work correctly on the *building-a-wagon* kind of problems either — they just get lucky. They do nothing to augment explanation structures during generalization. It just happens that to acquire a schema to build a wagon, *not* generalizing the explanation structure is the appropriate thing to do.

One can, of course, simply define the scope of EBL-type systems to exclude the *unstacking-to-move* concept and those like it. This is a mistake. First, the problem of augmenting the explanation during generalization, once seen, is ubiquitous. It is manifested in one form or another in most real-world domains. Second, if one simply defines the problem away, the resulting system could never guarantee that any of its concepts were as general as they should be. Even when such a system correctly constructed a concept like the *building-a-wagon* schema, it could not know that it had generalized properly. The system could not itself tell which concepts fall within its scope and which do not.

Observations of repeated application of a rule or operator may indicate that generalizing the number of rules in the explanation may be appropriate. However, alone this is insufficient. To be conducive to number generalization there must be a certain recursive structural pattern. That is, each application must achieve preconditions for the next. For example, consider stacking blocks. The same sort of repositioning of blocks occurs repeatedly, each building on the last. In this article, the vocabulary of predicate calculus is adopted to investigate this notion of structural recursion. The desired form of structural recursion is manifested as repeated application of an inference rule in such a manner that a portion of each consequent is used to satisfy some of the antecedents of the next application.

The next section introduces an implemented system designed to generalize the structure of explanations. Subsequent sections describe the algorithm used and illustrate it with a detailed example. Finally, before the conclusion, there are an empirical validation of the merits of

[1] The SOAR system (Laird *et al*, 1986) would seem to acquire a number of concepts which together are slightly more general. As well as a new operator for moving four blocks, the system would acquire new operators for moving three blocks, two blocks, and one block, but not for five or more.

generalizing the structure of explanations (including a comparison to the results of a standard EBL algorithm), a discussion of related work, and descriptions of several open research problems.

2. THE BAGGER SYSTEM

The **BAGGER** system (Building Augmented Generalizations by Generating Extended Recurrences) analyzes predicate calculus proofs and attempts to construct concepts that involve generalizing to N. Most of the examples under study use the situation calculus (McCarthy, 1963) to reason about actions, in the style of Green(1969). (Green's formulation is also discussed in (Nilsson, 1980).)

2.1. Situation Calculus

In situation calculus, predicates and functions whose values may change over time are given an extra argument which indicates the situation in which they are being evaluated. For example, rather than using the predicate *On(x,y)*, indicating that x is on y, the predicate *On(x,y,s)* is used, indicated that in situation s, x is on y. In this formulation, operators are represented as functions that map from one situation to another situation.

Problem solving with **BAGGER**'s situational calculus rules can be viewed as transforming and expanding situations until one is found in which the goal is known to be achieved. The **BAGGER** system has two types of inference rules: *inter-situational* rules which specify attributes that a new situation will have after application of a particular operator, and *intra-situational* rules which can embellish **BAGGER**'s knowledge of a situation by specifying additional conclusions that can be drawn within that situation.

Each inter-situational inference rule specifies knowledge about one particular operator. However, operators are not represented by exactly one inference rule. A major inference rule specifies most of the relevant problem-solving information about an operator. But it is augmented by many lesser inference rules which capture the operator's frame axioms and other facts about a new situation. This paradigm contrasts with the standard **STRIPS** (Fikes and Nilsson, 1971) formalism.[2] The inference rules of a **STRIPS**-like system are in a one-to-one correspondence with

[2] Fahlman (1974) and Fikes (1975) augmented the standard **STRIPS** model by allowing a distinction between primary and secondary relationships. Primary relationships are asserted directly by operators while secondary relationships are deduced from the primary ones as needed. While this serves the same purpose as **BAGGER**'s intra-situational rules, multiple inter-situational rules for an operator are not allowed (Waldinger,

the system's operators. Each inference rule fully specifies an operator's add- and delete-lists. These lists provide all of the changes needed to transform the current situation into the new situation. Any state not mentioned in an add- or delete-list is assumed to persist across the operator's application. Thus, the new situation is completely determined by the inference rule. In the **BAGGER** system this is not the case. Many separate inference rules are used to fully characterize the effect of an operator.

The advantage of the **STRIPS** approach is that the system can always be assured that it has represented all that there is to know about a new situation. However, this can also be a disadvantage. A **STRIPS**-like system must always muddle through all there is to know about a situation, no matter how irrelevant many facts may be to the current problem. Conversely, the advantages of **BAGGER**'s approach are that the inference rules are far less complex and therefore more manageable, the system's attention focussing is easier because it does not bog down in situations made overly-complex by many irrelevant facts, and a programmer can more easily write and update knowledge about operators. Furthermore, **STRIPS**-style operators do not allow disjunctive or conditional effects in their add- or delete-lists.

A potential disadvantage of **BAGGER**'s approach is that to completely represent the effects of applying an operator in a particular situation, the system must retrieve all of the relevant inference rules. However, this is not a task that arises in **BAGGER**'s problem solving. Indeed, there has been no attempt to guarantee the completeness of the system's inferential abilities. This means that there may be characteristics of a situation which **BAGGER** can represent but cannot itself infer.

2.2. Some Sample Problems

One problem solution analyzed by **BAGGER** is shown in figure 1. The goal is to place a properly-supported block so that its center is above the dotted line and within the horizontal confines of the line. **BAGGER** is provided low-level domain knowledge about blocks, including how to transfer a single block from one location to another and how to calculate its new horizontal and vertical position. Briefly, to move a block it must have nothing on it and there must be free space at which to place it. The system produces a situation calculus proof validating the actions shown in figure 1, in which three blocks must be moved to build the tower.

1977).

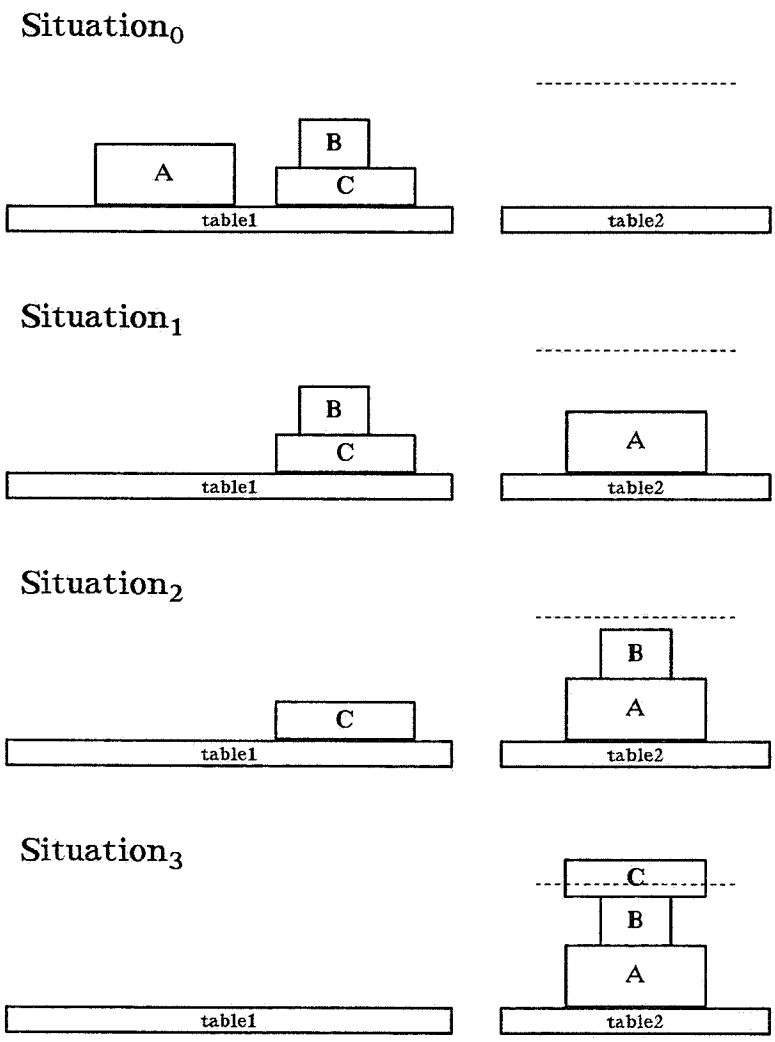

Figure 1. Constructing a Three-Block Tower

If a standard explanation-based generalization algorithm is applied to the resulting proof, a plan for moving *three* blocks will result. They need not be these same three blocks, any three distinct ones will suffice. Nor is it is necessary that the first block moved be placed on a table, any flat, clear surface is acceptable. Finally, the height of the tower need not be the same as that in the specific example. Given appropriately sized blocks, towers of any height can be constructed. Many characteristics of the problem are generalized. However, the fact that exactly three blocks are moved would remain.

If one considers the universe of all possible towers, as shown in figure 2, only a small fraction of them would be captured by the acquired rule. Separate rules would need to be learned for

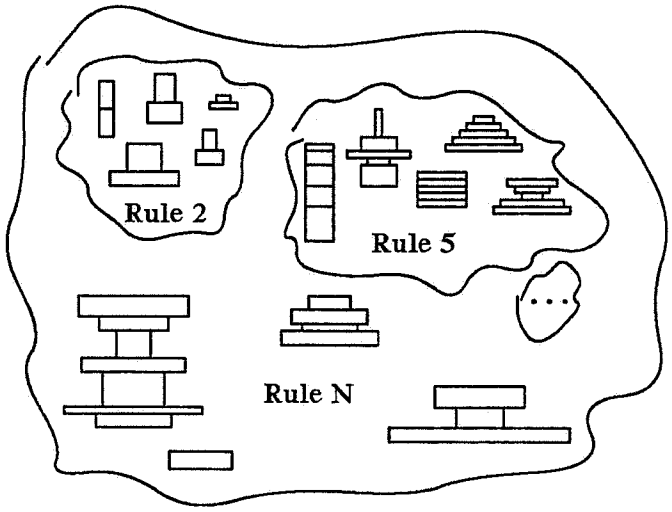

Figure 2. Universes of Constructible Towers

towers containing two blocks, five blocks, etc. What is desired is the acquisition of a rule that describes how towers containing any number of blocks can be constructed.

By analyzing the proof of the construction of the three-block tower, **BAGGER** acquires a general plan for building towers by stacking *arbitrary* numbers of blocks, as illustrated in figure 3. This new plan incorporates an indefinite number of applications of the previously known plan for moving a single block.

In another example, the system observes three blocks being removed from a stack in order to satisfy the goal of having a specific block be clear. Extending the explanation of these actions produces a plan for unstacking any number of blocks in order to clear a block within the stack. Figure 4 illustrates this general plan. The plan includes the system's realization that the last

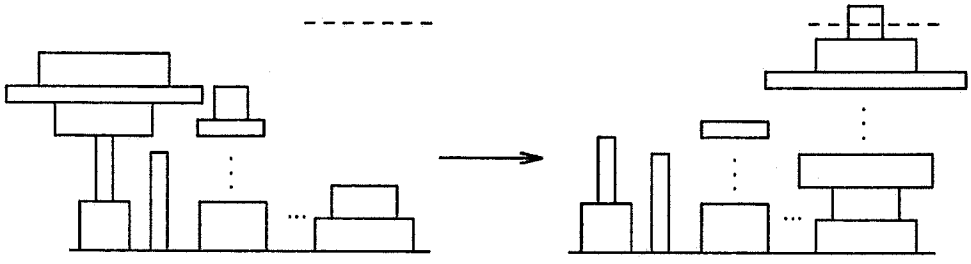

Figure 3. A General Plan for Constructing Towers

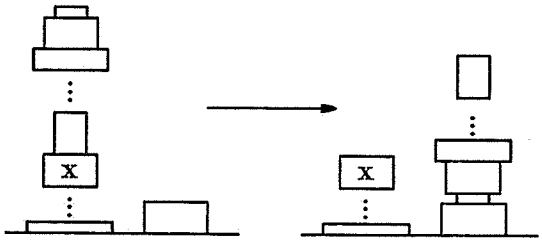

Figure 4. A General Plan for Unstacking Towers

unstacked block is currently clear and thus makes a suitable destination to place the next block to be moved. This knowledge is incorporated into the plan and no problem solving need be performed finding destinations once the first free location is found.

Unlike many other block-manipulation examples, in these examples it is *not* assumed that blocks can support only one other block. This means that moving a block does not necessarily clear its supporting block. Another concept learned by **BAGGER**, by observing two blocks being moved from on top another, is a general plan for clearing an object directly supporting any number of clear blocks. This plan is illustrated in figure 5.

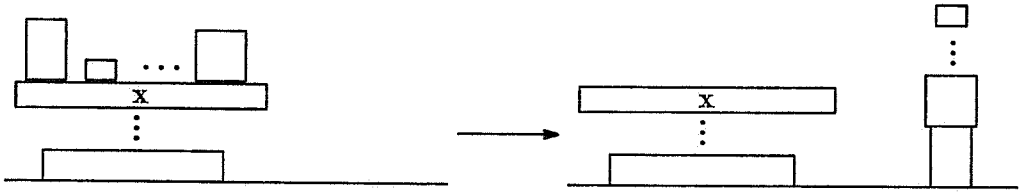

Figure 5. A General Plan for Clearing Objects

The domain of digital circuit design has also been investigated. By observing the repeated application of DeMorgan's law to implement two cascaded *AND* gates using *OR* and *NOT* gates, **BAGGER** produces a general version of DeMorgan's law which can be used to implement *N* cascaded *AND* gates with *N OR* and one *NOT* gate. This example, which does not use situation calculus, is shown in figure 6.

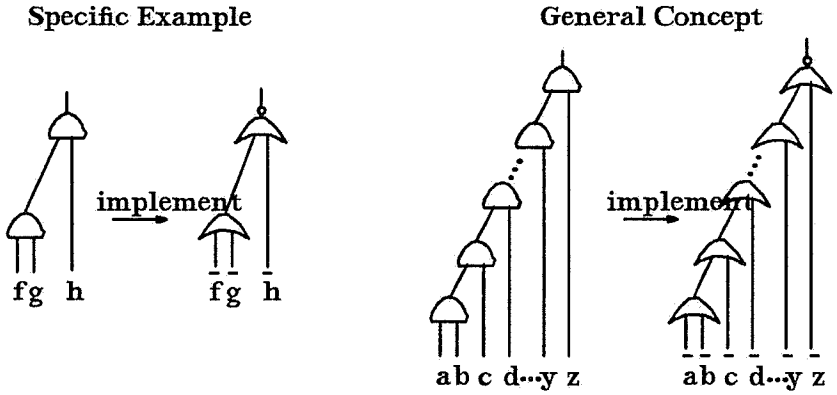

Figure 6. A Circuit Design Example

The next section presents the BAGGER generalization algorithm. Following that, there is a detailed presentation of the tower-building example, including the full proof tree and the acquired rule. The inference rules used in this example are described in the appendix. Complete details on the other examples, including the complete set of initial inference rules, the situation calculus proofs, and the acquired inference rules, can be found in (Shavlik, 1988).

3. GENERALIZATION IN BAGGER

Generalizing number, like more traditional generalization in EBL, results in the acquisition of a new inference rule. The difference is that the sort of rule that results from generalizing number describes the world after an indefinite number of world changes or other inferences have been made. Each such rule subsumes a potentially infinite class of standard situation calculus rules. Thus, with such rules the storage efficiency can be dramatically improved, the expressive power of the system is increased, and, as shown in section 5, the system's performance efficiency can also be higher than without these rules. This section describes how BAGGER generalizes number.

3.1. Sequential Rules

Like its standard inference rules, number-generalized rules in the BAGGER system are usually represented in situational calculus. In the previous section, two types of BAGGER inference rules are discussed: intra-situational rules and inter-situational rules. To define number-generalized rules, the inter-situational rules are further divided into two categories: simple inter-situational rules and sequential inter-situational rules (or simply sequential rules). Sequential rules apply a variable number of operators. Thus, within each application of a sequential rule many intermediate situations may be generated. The actual number of intermediate situations depends

on the complexity of the problem to be solved. The rule for building towers is an example of a sequential rule. This rule is able to construct towers of any number of blocks in order to achieve a specified goal height. The rule itself decides how many blocks are to be used and selects which blocks to use from among those present in the current situation.

Sequential rules, like their simple inter-situational counterparts, have an antecedent and a consequent. Also, like the simple versions, if the antecedent is satisfied, the consequent specifies properties of the resulting situation. Unlike the simple rules, the resulting situation can be separated from the initial situation by many operator applications and intermediate situations. For example, to build a tower, many block-moving operations must be performed. It is an important feature of sequential rules that no planning need be done in applying the intermediate operators. That is, if the antecedent of a sequential rule is satisfied, its entire sequence of operators can be applied without the need for individually testing or planning for the preconditions. The preconditions of each operator are guaranteed to be true by the construction of the sequential rule itself. Thus, the consequent of a sequential rule can immediately assert properties which must be true in the final situation. A sequential rule behaves much as a **STRIPS**-like macro-operator. It is termed a *sequential rule* and not a *macro-operator* because it is, in fact, a situational calculus rule and not an operator. It has a situation variable, does not specify *ADD* and *DELETE* lists, etc.

Sequential rules can be much more efficient than simply chaining together simple constituents. This improved efficiency is derived from three sources: 1) collecting together antecedents so that redundant and subsumed operator preconditions are eliminated, 2) heuristically ordering the antecedents, and, especially, 3) eliminating antecedents that test operator preconditions which, due to the structure of the rule, are known to be satisfied.

3.2. Representing Sequential Knowledge

A representational shift is crucial to this article's solution to the generalization to N problem. While objects in the world are represented within simple inference rules directly as predicate calculus variables, this is not possible for **BAGGER**'s sequential rules. A standard operator interacts with a known number of objects. Usually, this number is small. The rule representing the operator that moves blocks, for example, might take as arguments the block to be moved and the new location where it is to be placed. A simple inter-situational rule for this operator might specify that in the resulting situation, the block represented by the first argument is at the location specified by the second. This rule represents exactly one application of the move operator. There

are always two arguments. They can be conveniently represented by predicate calculus variables. That is, each of the world objects with which a simple operator interacts can be uniquely named with a predicate calculus variable. Sequential rules cannot uniquely name each of the important world objects. A rule for building towers must be capable of including an arbitrary number of blocks. The uninstantiated rule cannot know whether it is to be applied next to build a tower of five blocks, seven blocks, or 24 blocks. Since the individual blocks can no longer be named by unique variables within the rule, a shift is necessary to a scheme that can represent aggregations of world objects. Such a representational shift, similar to Weld's (1986), makes explicit attributes that are only implicitly present in the example. Thus, it shares many characteristics of constructive induction (Michalski, 1983; Rendell, 1985).

A new object called an *RIS* (for Rule Instantiation Sequence) is introduced to represent arbitrarily large aggregations of world objects. A sequential rule works directly with one of these generalized structures so that it need not individually name every world object with which it interacts. A sequential rule's *RIS* is constructed in the course of satisfying its antecedent. Once this is done, the *RIS* embodies all of the constraints required for the successive application of the sequence of operators that make up the plan.

3.3. The BAGGER Algorithm

Figure 7 schematically presents how **BAGGER** generalizes the structure of explanations. On the left is the explanation of a solution to a specific problem. In it, some inference rule is repeatedly applied a fixed number of times. In the generalized explanation, the number of applications of the rule is unconstrained. In addition, the properties that must hold in order to satisfy each application's preconditions, and to meet the antecedents in the goal, are expressed in terms of the initial situation. This means that portions of the explanation not directly involved in the chain of rule applications must also be expressed in terms of the initial state. When the initial situation has the necessary properties, the results of the new rule can be immediately determined, without reasoning about any of the intermediate situations.

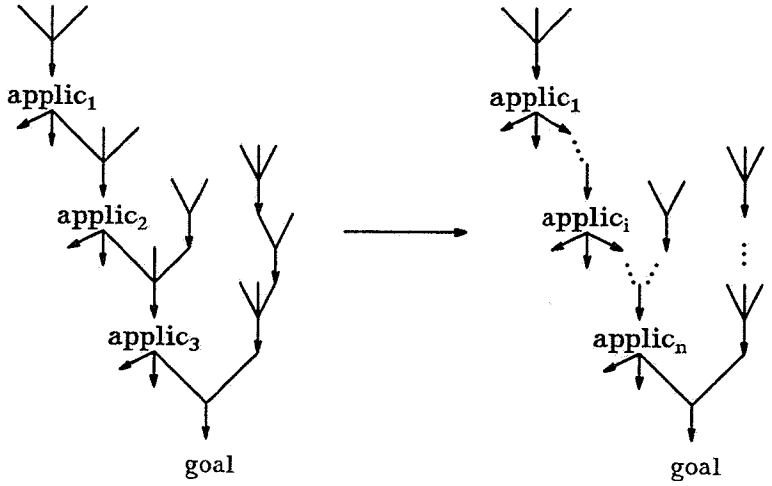

Figure 7. Generalizing the Structure of an Explanation

The generalization algorithm appears in figure 8. This algorithm is expressed in a pseudo-code, while the actual implementation is written in Lisp. The remainder of this section elaborates the pseudo-code. In the algorithm back arrows (←) indicate value assignment. The construct

for each *element* **in** *set* **do** *statement*

means that *element* is successively bound to each member of *set*, following which *statement* is evaluated. The functions *AddDisjunct* and *AddConjunct* alter their first argument. If either of *AddConjunct*'s arguments is *fail*, its answer is *fail*. *AddRule* places the new rule in the database of acquired rules.

The algorithm begins its analysis of a specific solution at the goal node. It then traces backward, looking for repeated rule applications. To be a candidate, some consequent of one instantiation of a rule must support the satisfication of an antecedent of another instantiation. These repeated applications need not directly connect — there can be intervening inference rules. Once a candidate is found, all the inter-connected instantiations of the underlying general rule are collected.

The general rule repeatedly applied is called a *focus rule*. After a focus rule is found, **BAGGER** ascertains how an *arbitrary* number of instantiations of this rule and any intervening rules can be concatenated together. This indefinite-length collection of rules is conceptually merged into the explanation, replacing the specific-length collection, and a new rule is produced from the augmented explanation.

```
procedure BuildNewBAGGERrule (goalNode)
  focusNodes ← CollectFocusRuleApplications(goalNode)
  antecedentsInitial ← BuildInitialAntecedents(Earliest(focusNodes))
  antecedentsIntermediate ← φ
  for each focusNode in focusNodes do
    answer ← ViewAsArbitraryApplic(focusNode, focusNodes)
    if answer ≠ fail then AddDisjunct(antecedentsIntermediate, answer)
  antecedentsFinal ← ViewAsArbitraryApplic(goalNode, focusNodes))
  consequents ← CollectGoalTerms(goalNode)
  if antecedentsIntermediate ≠ φ  ∧  antecedentsFinal ≠ fail
    then AddRule(antecedentsInitial, antecedentsIntermediate, antecedentsFinal, consequents)

procedure ViewAsArbitraryApplic (node, focusNodes)
  result ← φ
  for each antecedent in Antecedents(node) do
    if Axiom?(antecedent) then true
    else if SupportedByEarlierNode?(antecedent, focusNodes) then
      AddConjunct(result, CollectNecessaryEqualities(antecedent, Supporter(antecedent)))
    else if SituationIndependent?(antecedent) then AddConjunct(result, antecedent)
    else if SupportedByPartiallyUnwindableRule?(antecedent) then
      AddConjunct(result, CollectResultsOfPartiallyUnwinding(antecedent))
      AddConjunct(result, ViewAsArbitraryApplic(PartiallyUnwind(antecedent), focusNodes))
    else if SupportedByUnwindableRule?(antecedent) then
      AddConjunct(result, CollectResultsOfUnwinding(antecedent))
    else if SupportedByRuleConsequent?(antecedent) then
      AddConjunct(result, CollectNecessaryEqualities(antecedent, Supporter(antecedent)))
      AddConjunct(result, ViewAsArbitraryApplic(SupportingRule(antecedent), focusNodes))
    else return fail
  return result
```

Figure 8. The BAGGER Generalization Algorithm

A specific solution contains several instantiations of the general rule chosen as the focus rule.

Each of these applications of the rule addresses the need of satisfying the rule's antecedents, possibly in different ways. For example, when clearing an object, the blocks moved can be placed in several qualitatively different types of locations. The moved block can be placed on a table (assuming the domain model specifies that tables always have room), it can be placed on a block moved in a previous step, or it can be placed on a block that was originally clear.

BAGGER analyzes all applications of the general focus rule that appear in the specific example. When several instantiations of the focus rule provide sufficient information for different generalizations, **BAGGER** collects the preconditions for satisfying the antecedents of each in a disjunction of conjunctions (one conjunct for each acceptable instantiation). Common terms are factored out of the disjunction. If none of the instantiations of the focus rule provide sufficient information for generalizing the structure of the explanation, no new rule is learned by **BAGGER**.

Three classes of terms must be collected to construct the antecedents of a new rule. First, the antecedents of the initial rule application in the arbitrary length sequence of rule applications must be satisfied. To do this, the antecedents of the focus rule are used. Second, the preconditions imposed by chaining together an arbitrary number of rule applications must be collected. These are derived by analyzing each inter-connected instantiation of the focus rule in the sample proof. Those applications that provide enough information to be viewed as the arbitrary *ith* application produce this second class of preconditions. Third, the preconditions from the rest of the explanation must be collected. This determines the constraints on the final applications of the focus rule.

In order to package a sequence of rule applications into a single sequential rule, the preconditions that must be satisfied at each of the N rule applications must be collected and combined. The preconditions for applying the resulting extended rule must be specifiable in terms of the initial state, and *not* in terms of intermediate states. This insures, given that the necessary conditions are satisfied in the initial state, a plan represented in a sequential rule will run to completion without further problem solving, regardless of the number of intervening states necessary. For example, there is no possibility that a plan will lead to moving N-2 blocks and then get stuck. If the preconditions for the *ith* rule application were expressed in terms of the result of the *(i—1)th* application, each of the N rule applications would have to be considered in turn to see if the preconditions of the next are satisfied. This is not acceptable. In the approach taken, extra work during generalization and a possible loss of generality are traded off for a rule whose

preconditions are easier to check.

When a focus rule is concatenated an arbitrary number of times, variables need to be chosen for each rule application. The *RIS*, a sequence of p-dimensional *vectors*, is used to represent this information. The general form of the *RIS* is:

$$<v_{1,1}, \ldots, v_{1,p}>, <v_{2,1}, \ldots, v_{2,p}>, \ldots, <v_{n,1}, \ldots, v_{n,p}> \quad (1)$$

In the tower-building example of figure 1, initially $p = 3$: the current situation, the object to be moved, and the object upon which the moved object will be placed.

Depending on the rule used, the choice of elements for this sequence may be constrained. For example, certain elements may have to possess various properties, specific relations may have to hold among various elements, some elements may be constrained to be equal to or unequal to other elements, and some elements may be functions of other elements. Often choosing the values of the components of one vector, determines the values of components of subsequent vectors. For instance, when building a tower, choosing the block to be moved in step i also determines the location to place the block to be moved in step $i+1$.

To determine the preconditions in terms of the initial state, each of the focus rule instantiations appearing in the specific proof is viewed as an arbitrary (or *ith*) application of the underlying rule. The antecedents of this rule are analyzed as to what must be true of the initial state in order that it is guaranteed the *ith* collection of antecedents are satisfied when needed. This involves analyzing the proof tree, considering how each antecedent is proved. An augmented version of a standard explanation-based generalization algorithm (Mooney and Bennett, 1986) is used to determine which variables in this portion of the proof tree are constrained in terms of other variables.

Once this is done, the variables are expressed as components of the p-dimensional vectors described above, and the system ascertains what must be true of this sequence of vectors so that each antecedent is satisfied when necessary. All antecedents of the chosen instantiation of the focus rule must be of one of the following types for generalizing to N to be possible:

(1) The antecedent may be an *axiom*. Since an axiom always holds, it need not appear as a precondition in the final rule.

(2) The antecedent may be supported by a consequent of an earlier application of the focus rule. Terms of this type place inter-vector constraints on the sequence of p-dimensional vectors. These constraints are computed by unifying the general versions of the two terms.

(3) The antecedent may be *situation-independent*. Terms of this type are unaffected by actions.

(4) The antecedent may be supported by an "unwindable" or partially "unwindable" rule. When this happens, the antecedent is unwound to an arbitrary earlier state and all of the preconditions necessary to insure that the antecedent holds when needed are collected. A *partially unwindable* rule goes back an indefinite number of situations, from which the algorithm continues recursively. If no other inference rules are in the support of the unwindable rule, then it is unwound all the way to the initial state. The process of unwinding is further elaborated later. It, too, may place inter-vector constraints on the sequence of p-dimensional vectors.

(5) The antecedent is supported by other terms that are satisfied in one of the above ways. When traversing backwards across a supported antecedent, the system collects any inter-vector constraints produced by unifying the general version of the antecedent with the general version of the consequent that supports it.

Notice that antecedents are considered satisfied when they can be expressed in terms of the initial state, and *not* when a leaf of the proof tree is reached. Conceivably, to satisfy these antecedents in the initial state could require a large number of inference rules. If that is the case, it may be better to trace backwards through these rules until more *operational* terms are encountered. This *operationality/generality* trade-off (DeJong and Mooney, 1986; Keller, 1987; Mitchell, Keller, and Kedar-Cabelli, 1986; Segre, 1987; Shavlik, DeJong, and Ross, 1987) is a major issue in explanation-based learning, but will not be discussed further here. Usually the cost of increased operationality is more limited applicability. An empirical analysis of the effect of this trade-off in the **BAGGER** system appears in (Shavlik, 1988).

A second point to notice is that not all proof subtrees will terminate in one of the above ways. If this is the case, this application of the focus rule cannot be viewed as an arbitrary *ith* application.[3]

---

[3] An alternative approach to this would be to have the system search through its collection of unwindable rules and incorporate a relevant one into the proof structure. To study the limits of this article's approach to

The possibility that a specific solution does not provide enough information to generalize to N is an important point in explanation-based approaches to generalizing number. A concept involving an arbitrary number of substructures may involve an arbitrary number of substantially different problems. Any specific solution will only have addressed a finite number of these sub-problems. Due to fortuitous circumstances in the example some of the potential problems may not have arisen. To generalize to N, a system must recognize all the problems that exist in the *general* concept and, by analyzing the specific solution, surmount them. Inference rules of a certain form (described later) elegantly support this task in the **BAGGER** system. They allow the system to reason backwards through an arbitrary number of actions.

Figure 9 illustrates how consequents of an earlier application of a focus rule can satisfy some antecedents of a later instantiation. This figure contains a portion of the proof for the tower-building example. (The full proof tree is presented and discussed later.) Portions of two

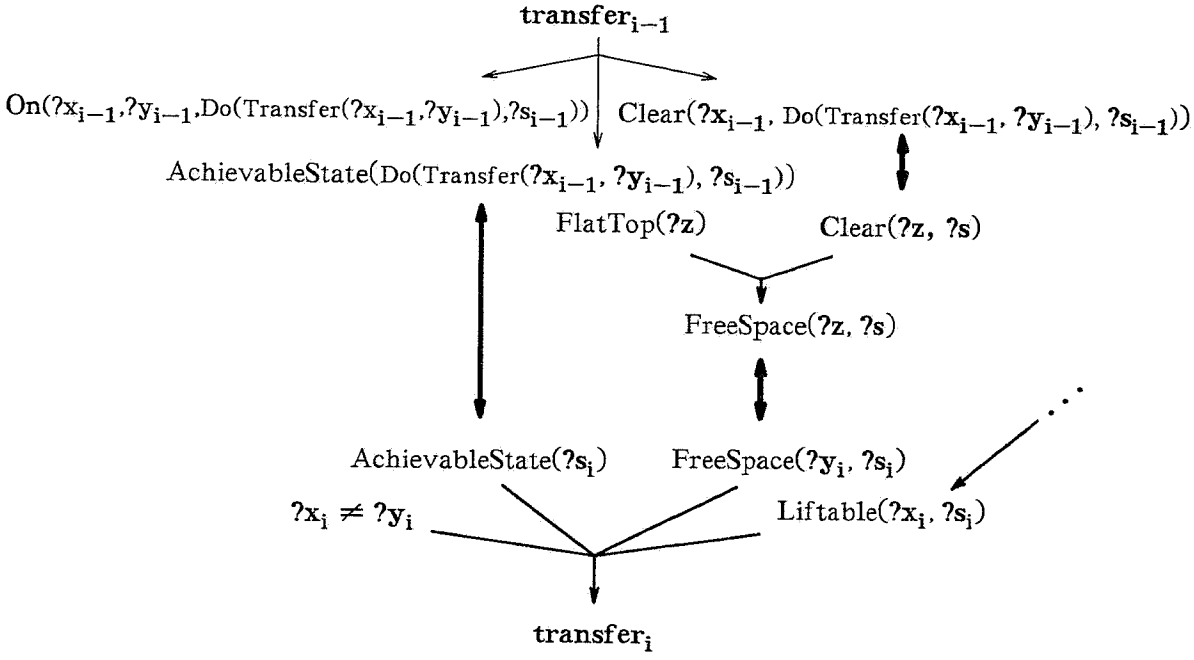

Figure 9. Satisfying Antecedents by Previous Consequents

---

generalizing to N, it is required that *all* necessary information be present in the explanation; no problem-solving search is performed during generalization. Another approach would be to assume the problem solver could overcome this problem at rule application time. This second technique, however, would eliminate the property that a learned plan will always run to completion whenever its preconditions are satisfied in the initial state.

consecutive transfers are shown. All variables are universally quantified. Arrows run from the antecedents of a rule to its consequents. Double-headed arrows represent terms that are equated in the specific explanation. The generalization algorithm enforces the unification of these paired terms, leading to the collection of equality constraints.

There are four antecedents of a transfer. To define a transfer, the block moved (x), the object on which it is placed (y), and the current state (s) must be specified, and the constraints among these variables must be satisfied. One antecedent, the one requiring a block not be placed on top of itself, is type 3 — it is *situation-independent*. The next two antecedents are type 2. Two of the consequents of the *(i—1)th* transfer are used to satisfy these antecedents of the *ith* transfer. During $transfer_{i-1}$, in state s_{i-1} object x_{i-1} is moved on to object y_{i-1}. The consequents of this transfer are that a new state is produced, the object moved is clear in the new state, and x_{i-1} is on y_{i-1} in the resulting state.

The state that results from $transfer_{i-1}$ satisfies the second antecedent of $transfer_i$. Unifying these terms defines s_i in terms of the previous variables in the *RIS*.

Another antecedent requires that, in state s_i, there be space on object y_i to put block x_i. This antecedent is type 5, and, hence, the algorithm traverses backwards through the rule that supports it. An inference rule specifies that a clear object with a flat top has free space. The clearness of x_{i-1} after $transfer_{i-1}$ is used. Unifying this collection of terms leads, in addition to the redundant definition of s_i, to the equating of y_i with z and x_{i-1}. This means that the previously moved block always provides a clear spot to place the current block, which leads to the construction of a tower.

The fourth antecedent, that x_i be liftable, is also type 5. A rule (not shown) states that an object is liftable if it is a clear block. Block x_i is determined to be clear because it is clear in the initial state and nothing has been placed upon it. Tracing backwards from the liftable term leads to several situation-independent terms and the term $Supports(?x_i, \phi, ?s_i)$. Although this term contains a situation variable, it is satisfied by an "unwindable rule," and is type 4.

Equation 2 presents the form required for a rule to be unwindable. The consequent must match one of the antecedents of the rule. Hence, the rule can be applied recursively. This feature is used to "unwind" the term from the *ith* state to an earlier state, often the initial state. Occasionally there can be several unwindable rules in a support path. For example, a block might support another block during some number of transfers, be cleared, remain clear during another

sequence of transfers, and finally be added to a tower. The variables in the rule are divided into three groups. First, there are the x variables. These appear unchanged in both the consequent's term **P** and the antecedent's term **P**. Second, there are the y variables which differ in the two **P**'s and the z variables that only appear in the antecedents. Finally, there is the state variable (s). There can be additional requirements of the x, y, and z variables (via predicate **Q**), however, these requirements cannot depend on a state variable.

Applying equation 2 recursively produces equation 3. This rule determines the requirements on the earlier state so that the desired term can be guaranteed in state i. Except for the definition of the next state, none of the antecedents depends on the intermediate states. Notice that a collection of y and z variables must be specified. Any of these variables not already contained in the *RIS* are added to it. Hence, the *RIS* is also used to store the results of intermediate computations. Since the predicate **Q** does not depend on the situation, it can be evaluated in the initial state.

$$\begin{aligned}
&\mathrm{P}\,(x_{i,1}, \ldots, x_{i,\mu}, y_{i-1,1}, \ldots, y_{i-1,\nu}, s_{i-1}) \\
&\qquad \text{and} \\
&\mathrm{Q}\,(x_{i,1}, \ldots, x_{i,\mu}, y_{i-1,1}, \ldots, y_{i-1,\nu}, y_{i,1}, \ldots, y_{i,\nu}, \ldots, z_{i,1}, \ldots, z_{i,\omega}) \\
&\qquad \text{and} \\
&s_i = Do\,(x_{i,1}, \ldots, x_{i,\mu}, y_{i-1,1}, \ldots, y_{i-1,\nu}, , \ldots, z_{i,1}, \ldots, z_{i,\omega}, s_{i-1}) \\
&\quad \longrightarrow \\
&\mathrm{P}\,(x_{i,1}, \ldots, x_{i,\mu}, y_{i,1}, \ldots, y_{i,\nu}, s_i)
\end{aligned} \tag{2}$$

$$\begin{aligned}
&\mathrm{P}\,(x_{i,1}, \ldots, x_{i,\mu}, y_{j,1}, \ldots, y_{j,\nu}, s_1) \text{ and } 0 < \mathrm{j} < \mathrm{i} \\
&\qquad \text{and} \\
&\forall\, k \in j+1, \ldots, i \\
&\qquad \mathrm{Q}\,(x_{i,1}, \ldots, x_{i,\mu}, y_{k-j,1}, \ldots, y_{k-j,\nu}, y_{k,1}, \ldots, y_{k,\nu}, \ldots, z_{k,1}, \ldots, z_{k,\omega}) \\
&\qquad\qquad \text{and} \\
&\qquad s_k = Do\,(x_{i,1}, \ldots, x_{i,\mu}, y_{k-j,1}, \ldots, y_{k-j,\nu}, , \ldots, z_{k-j,1}, \ldots, z_{k-j,\omega}, s_{k-1}) \\
&\longrightarrow \\
&\mathrm{P}\,(x_{i,1}, \ldots, x_{i,\mu}, y_{i,1}, \ldots, y_{i,\nu}, s_i)
\end{aligned} \tag{3}$$

The requirements on the predicate **Q** are actually somewhat less restrictive. Rather than requiring this predicate to be situation-independent, all that is necessary is that any term containing a situation argument be supported (possibly indirectly) by an application of a focus rule. The important characteristic is that the satisfication of the predicate **Q** can be specified in terms of the initial situation only. Separately unwinding a predicate **Q** while in the midst of unwinding a predicate **P** is not possible with the current algorithm, and how this can be accomplished is an open research issue.

Frame axioms often satisfy the form of equation 2. Figure 10 shows one way to satisfy the need to have a clear object at the *ith* step. Assume the left-hand side of figure 10 is a portion of some proof. This explanation says block x_i is clear in state s_i because it is clear in state s_{i-1} and the block moved in $transfer_{i-1}$ is not placed upon x_i. Unwinding this rule leads to the result that block x_i will be clear in state s_i if it is clear in state s_1 and x_i is never used as the new support block in any of the intervening transfers.

To classify an instantiation of a rule as being unwindable, the rule must be applied at least *twice* successively. This heuristic prevents generalizations that are likely to be spurious. Just like when looking for multiple applications of the focus rule, multiple applications are required for unwindable rules. The intent of this is to increase the likelihood that a generalization is being made that will be prove useful in the future. For example, imagine some rule represents withdrawing some money from a bank and also imagine this rule is of the form of equation.2. Assume that in state 5, John withdraws $500 to buy a television, while in states 1-4, the amount of money he has in the bank is unaffected. While it is correct to generalize this plan to include any number of trips to the bank in order to get sufficient money for a purchase, it does not seem proper to do so.

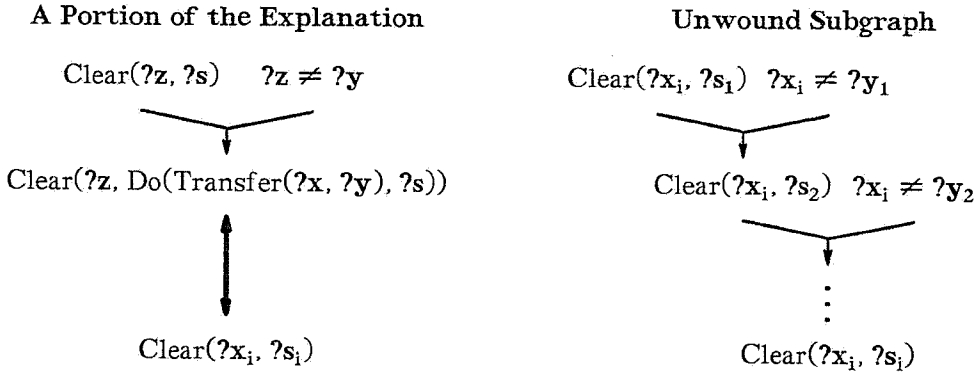

Figure 10. Unwinding a Rule

Rather, the generalization should be to a single trip to the bank at *any* time. Frame axioms are exceptions to this constraint - they only need to be applied once to be considered unwindable. Since frame axioms only specify what remains unchanged, there is no risk in assuming an arbitrary number of successive applications.

Once the repeated rule portion of the extended rule is determined, the rest of the explanation is incorporated into the final result. This is accomplished in the same manner as the way antecedents are satisfied in the repeated rule portion. The only difference is that the focus rule is now viewed as the *Nth* rule application. As before, antecedents must be of one of the five specified types. If all the terms in the goal cannot be satisfied in the arbitrary *Nth* state, no rule is learned.

The consequents of the final rule are constructed by collecting those generalized final consequents of the explanation that directly support the goal.

Even though all the antecedents of a sequential **BAGGER** rule are evaluated in the initial state, substantial time can be spent finding satisfactory bindings for the variables in the rule. Simplifying the antecedents of a rule acquired using EBL can increase the efficiency of the rule (Minton, Carbonell, Etzioni, Knoblock, and Kuokka, 1987; Prieditis and Mostow; 1987). After a rule is constructed by the **BAGGER** generalization algorithm, duplicate antecedents are removed and the remainder are rearranged by the system in an attempt to speed-up the process of satisfying the rule. This involves several processes. Heuristics are used to estimate whether is better to construct sequences from the first vector forward or from the last vector backward. Terms not effected by the intermediate antecedent are moved so that they are tested as soon as possible. Terms involving arithmetic are placed so that all their arguments are bound when they are evaluated. Finally, within each grouping, antecedents are arranged so that terms involving the same variable are near each other.

The next section discusses the sequential-rule produced by this algorithm when applied to the problem of building a tower.

4. DETAILS OF THE STACKING EXAMPLE

This section presents the details of one of **BAGGER**'s sequential rules. The proof that explains the tower-building actions of figure 1 appears in figure 11. This graph is produced by the **BAGGER** system, however nodes have been rearranged by hand for the sake of readability. Since the situation arguments are quite lengthy, they are abbreviated and a key appears in the figure.

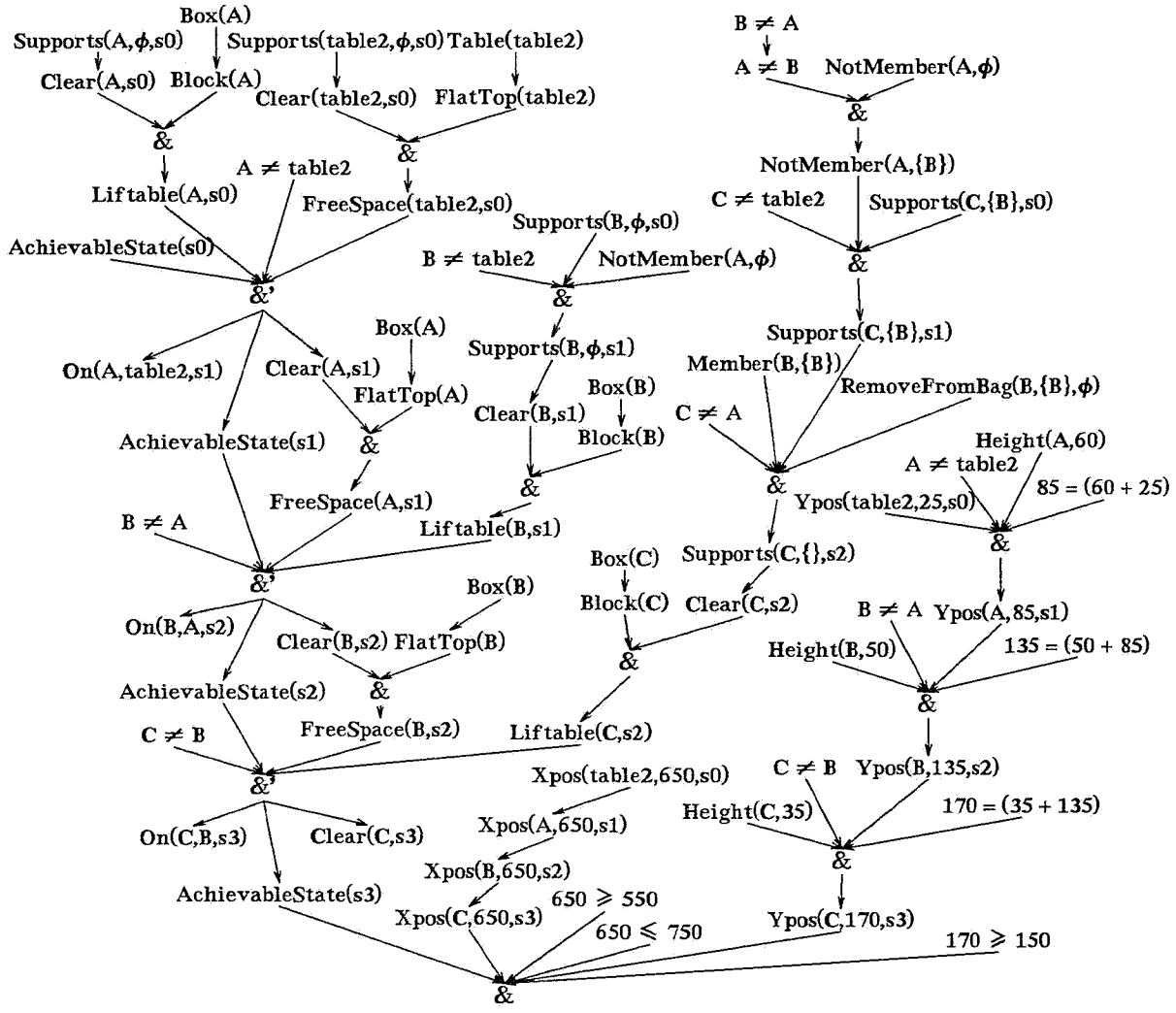

Figure 11. Situation Calculus Plan for Stacking Three Blocks

Abbreviation Key

s0	*the initial state*	s2	Do(Transfer(B,A),Do(Transfer(A,table2),s0))
s1	Do(Transfer(A,table2),s0)	s3	Do(Transfer(C,B),Do(Transfer(B,A),Do(Transfer(A,table2),s0)))

Arrows run from the antecedent of a rule to its consequent. When a rule has multiple antecedents or consequents, an ampersand (&) is used. Descriptions of all the rules used in this structure are contained in the appendix. The primed ampersands are the instantiations of the focus rule, while

the lowest ampersand is the goal node.

The goal provided to the backward-chaining theorem prover that produced this graph is:

$$\exists \quad \text{AchievableState(?state)} \quad \wedge$$
$$\text{Xpos(?object, ?px, ?state)} \wedge \text{?px} \geqslant 550 \wedge \text{?px} \leqslant 750 \wedge$$
$$\text{Ypos(?object, ?py, ?state)} \wedge \text{?py} \geqslant 150.$$

This says that the goal is to prove the existence of an achievable state, such that in that state the horizontal position of some object is between 550 and 750 and the vertical position of that same objects is at least 150.

The sequential-rule produced by analyzing this explanation structure appears in table 1. The remainder of this section describes how each term in this table is produced. Line numbers have been included for purposes of reference. For readability, the new rule is broken down into components, as shown in equation 4. While **BAGGER**'s reordering of a new rule's antecedents means the presented rule is somewhat harder to read, table 1 accurately reflects the rule acquired and used by the system.

4.1. Producing the Initial Antecedents

The initial antecedents in the first line of the rule establish a sequence of vectors, the initial state, and the first vector contained in the sequence. Subscripts are used to indicate components of vectors, as a shorthand for functions that perform this task. For example, $?v_{1,3}$ is shorthand for *ThirdComponent* $(?v_1)$. Lines 2 and 3 contain the antecedents of the first application in the chain of applications. These are the same terms that appear in the focus rule (the first rule in table A.2), except that the components of v_1 are used. The system has knowledge of which arguments are situation variables and the initial state constant *s0* is placed in these positions. The other terms in this grouping are produced by the unwinding process (*Height* , *Xpos* , *Ypos* , and the addition term) or are moved ($\geqslant$ and $\leqslant$) from the final antecedents to the initial antecedents because their variables are not influenced by the intermediate antecedents. The terms produced by unwinding are described further in what follows.

4.2. Analyzing the Applications of the Focus Rule

Lines 5-11 contain the preconditions derived by analyzing the three instantiations of the focus rule. In this implication, v_i - an arbitrary vector in the sequence (other than the first) - is used, as these constraints must be satisfied for each of the applications that follow the first. Vector v_{i-1} is

Table 1 The Components of the Learned Rule

Antecedents $_{initial}$

(1) Sequence(?seq) ∧ InitialVector(?v_1,?seq) ∧ State(s0) ∧ ?$v_{1,1}$ = s0 ∧

(2) FreeSpace(?$v_{1,3}$, s0) ∧ Liftable(?$v_{1,2}$, s0) ∧ Height(?$v_{1,2}$, ?$v_{1,4}$) ∧

(3) Xpos(?$v_{1,3}$, ?px, s0) ∧ Ypos(?$v_{1,3}$, ?new, s0) ∧ ?$v_{1,2}$ ≠ ?$v_{1,3}$ ∧

(4) ?$v_{1,5}$ = (?$v_{1,4}$ + ?new) ∧ ?px ≥ ?xmin ∧ ?px ≤ ?xmax

Antecedent $_{intermediate}$

(5) [Member(?v_i, ?seq) ∧ ?v_i ≠ ?v_1 ∧ Member(?v_{i-1}, ?seq) ∧ Predecessor(?v_{i-1}, ?v_i, ?seq)
→

(6) ?$v_{i,3}$ = ?$v_{i-1,2}$ ∧ ?$v_{i,1}$ = Do(Transfer(?$v_{i-1,2}$, ?$v_{i-1,3}$), ?$v_{i-1,1}$) ∧ FlatTop(?$v_{i,3}$) ∧

(7) Block(?$v_{i,2}$) ∧ Height(?$v_{i,2}$, ?$v_{i,4}$) ∧ ?$v_{i,2}$ ≠ ?$v_{i,3}$ ∧ ?$v_{i,5}$ = (?$v_{i,4}$ + ?$v_{i-1,5}$)

(8) [[[Member(?v_j, ?seq) ∧ Earlier(?v_j, ?v_i, ?seq) → ?$v_{i,2}$ ≠ ?$v_{j,3}$] ∧ Supports(?$v_{i,2}$, ϕ, s0)]

(9) ∨ [[Member(?v_j, ?seq) ∧ Earlier(?v_j, ?v_{i-1}, ?seq) → NotMember(?$v_{j,2}$, {?$v_{i-1,2}$})] ∧

(10) [Member(?v_j, ?seq) ∧ Earlier(?v_j, ?v_{i-1},?seq) → ?$v_{i,2}$ ≠ ?$v_{j,3}$] ∧

(11) Supports(?$v_{i,2}$, {?$v_{i-1,2}$}, s0) ∧ ?$v_{i,2}$ ≠ ?$v_{i-1,3}$]]]

Antecedents $_{final}$

(12) FinalVector(?v_n,?seq) ∧ ?py = ?$v_{n,5}$ ∧ ?state = Do(Transfer(?$v_{n,2}$, ?$v_{n,3}$), ?$v_{n,1}$) ∧

(13) ?object = ?$v_{n,2}$ ∧ ?py ≥ ?ymin

Consequents

(14) State(?state) ∧ Xpos(?object, ?px, ?state) ∧ ?px ≤ ?xmax ∧ ?px ≥ ?xmin ∧

(15) Ypos(?object, ?py, ?state) ∧ ?py ≥ ?ymin

This rule extends sequences 1 → N.

$$Antecedents_{initial} \wedge Antecedent_{intermediate} \wedge Antecedents_{rest} \rightarrow Consequents \qquad (4)$$

the vector immediately preceding v_i. It is needed because some of the antecedents of the *ith* application are satisfied by the *(i—1)th* application. Although some preconditions in the new rule

involve v_i and v_{i-1}, these preconditions all refer to conditions in the initial state. They do *not* refer to results in intermediate states.

The final two of the three instantiations of the focus rule produce sufficient information to determine how the antecedents of the rule can be satisfied in the *ith* application. In the first application (upper left of figure 11), neither the support for *Liftable* nor the support for *FreeSpace* provide enough information to determine the constraints on the initial state so that these terms can be satisfied in an arbitrary step. In both cases, the proof only had to address clearness in the current state. No information is provided within the proof as to how clearness can be guaranteed to hold in some later state.

The two other instantiations of the focus rule provide sufficient information for generalization. Two different ways of satisfying the antecedents are discovered, and, hence, a disjunction is learned. The common terms in these two disjuncts appear in lines 6 and 7, while the remaining terms for the first disjunction are in line 8 and for the second in lines 9-11.

The third term in line 7 is the vector form of the inequality that is one of the antecedents of the focus rule. This, being situation-independent, is a type 3 antecedent. In vector form, it becomes $v_{i,2} \neq v_{i,3}$. It constrains possible collections of vectors to those that have different second and third members. This constraint stems from the requirement that a block cannot be stacked on itself.

Both of the successful applications of the focus rule have their *AchievableState* term satisfied by a consequent of a previous application. These terms are type 2 and require collection of the equalities produced by unifying the general versions of the matching consequents and antecedents. (See figure 9 for the details of these matchings.) The equality that results from these unifications is the second term of line 6. Thus, the next state is always completely determined by the previous one. No searching needs to be done in order to choose the next state. (Actually, no terms are ever evaluated in these intermediate states. The only reason they are recorded is so that the final state can be determined, for use in setting the situation variable in the consequents.)

Both successful applications have their *FreeSpace* term satisfied in the same manner. Traversing backwards across one rule leads to a situation independent term (*FlatTop* - line 6) and the consequent of an earlier application (*Clear*). Unifying the two clear terms (again, see figure 9) produces the first two equalities in line 6. This first equality means that the block to be moved in

the *ith* step can always be placed on top of the block to be moved in the *(i—1)th* step. No problem solving need be done to determine the location at which to continue building the tower.

The *Block* term in line 7 is produced during the process of analyzing the way the *Liftable* term is satisfied. The remaining portion of the analysis of *Liftable* produces the terms in the disjunctions. As in the initial antecedents, the *Height* and addition terms in line 7 are produced during the analysis of the terms in the goal, which is described later.

In the second application of the focus rule, which produces the first disjunct, a clear block to move is acquired by finding a block that is clear because it supports nothing in the initial state and nothing is placed on it later. The frame axiom supporting this is an unwindable rule. Unwinding it to the initial state produces line 8. The *Supports* term must hold in the initial state and the block to be moved in step i can never be used as the place to locate a block to be moved in an earlier step. The general version of the term *NotMember* (A, ϕ) does not appear in the learned rule because it is an axiom that nothing is a member of the empty set. (An earlier unification, from the rule involving *Clear*, requires that the second variable in the general version of *NotMember* term be ϕ.)

Notice that this unwinding restricts the applicability of the acquired rule. The first disjunct requires that if an initially clear block is to be added to the tower, nothing can ever be placed on it, even temporarily. A more general plan would be learned, however, if in the specific example a block is temporarily covered. In that case, in the proof there would be several groupings of unwindable rules; for awhile the block would remain clear, something would then be placed on it and it would remain covered for several steps, and finally it would be cleared and remain that way until moved. Although this clearing and unclearing can occur repeatedly, the current **BAGGER** algorithm is unable to generalize number within unwindable subproofs.

The second disjunct (lines 9-11) results from the different way a liftable block is found in the third application of the focus rule. Here a liftable block is found be using a block that initially supported one other block, which is moved in the previous step, and where nothing else is moved to the lower block during an earlier rule application. Unwinding the subgraph for this application leads to the requirements that initially one block is on the block to be moved in step i, that block be moved in step *(i—1)*, and nothing else is scheduled to be moved to the lower block during an earlier rule applications. Again, some terms do not appear in the learned rule (*Member* and *RemoveFromBag*) because, given the necessary unifications, they are axioms. This time

NotMember is not an axiom, and hence, appears. If the specific example were more complicated, the acquired rule would reflect the fact that the block on top can be removed in some *earlier* step, rather than necessarily in the *previous* step.

4.3. Analyzing the Rest of the Explanation

Once all of the instantiations of the focus rule are analyzed, the goal node is visited. This produces lines 12 and 13, plus some of the earlier terms.

The *AchievableState* term of the goal is satisfied by the final application of the focus rule, leading to the third term in line 12.

The final *X*-position is calculated using an unwindable rule. Tracing backwards from the *Xpos* in the goal to the consequent of the unwindable rule produces the first term in line 13, as well as the third term in line 12. When this rule is unwound it produces the first term of line 3 and the second term of line 6. Also, matching the *Xpos* term in the antecedents with the one in the consequents, so that equation 2 applies, again produces the first term in line 6. Since there are no "Q"-terms (equation 2), no other preconditions are added to the intermediate antecedent.

The inequalities involving the tower's horizontal position are state-independent; their general forms are moved to the initial antecedents because their arguments are not effected by satisfying the intermediate antecedent. These terms in the initial antecedents involving *?px* insure that the tower is started underneath the goal.

Unwindable rules also determine the final *Y*-position. Here "Q"-terms are present. The connection of two instantiations of the underlying general rule appears in figure 12. This general rule is unwound to the initial state, which creates the second term of line 3 and the second term of line 6. The last three terms of line 7 are also produced, as the "Q"-terms must hold for each application of the unwound rule. This process adds two components to the vectors in the *RIS*. The first ($?v_{i,4}$) comes from the *?hx* variable, which records the height of the block being added to the tower. The other ($?v_{i,5}$) comes from the variable *?yPos* 2. It records the vertical position of the block added, and hence, represents the height of the tower. The *?ypos* variable does not lead to the creation of another *RIS* entry because it matches the *?yPos* 2 variable of the previous application. All that is needed is a *?ypos* variable for the first application. Similarly, matching the *Ypos* term in the antecedents with the one in the consequents produces the first term in line 6.

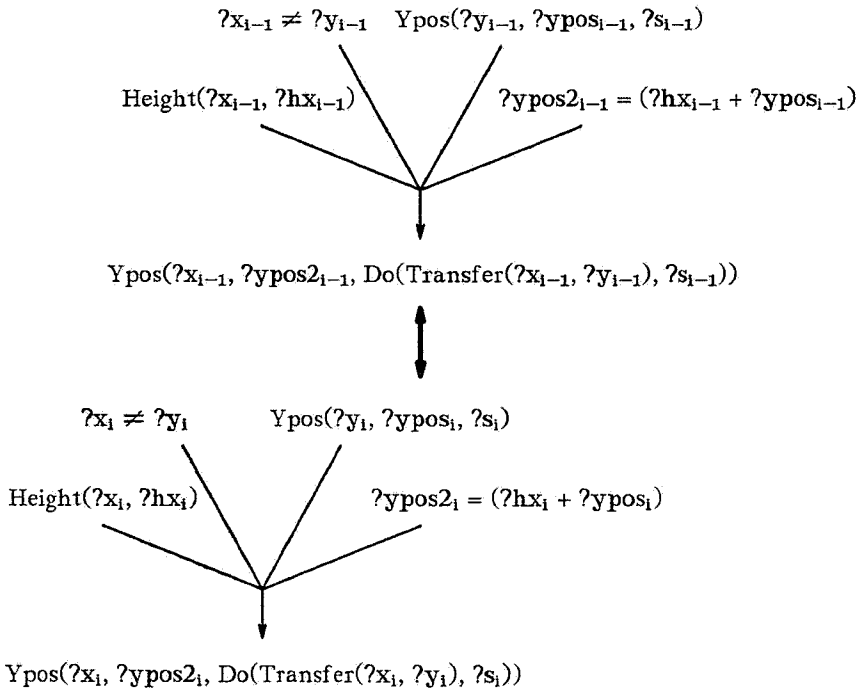

Figure 12. Calculating the Vertical Position of the *i*th Stacked Block

The last conjunct in the goal produces the second term on line 13. This precondition insures that the final tower is tall enough.

Finally, the general version of the goal description is used to construct the consequents of the new rule (lines 14 and 15).

5. EMPIRICAL ANALYSIS

An empirical analysis of the performance of the **BAGGER** system is presented in this section. This system is compared to an implementation of a standard explanation-based generalization algorithm (Mooney and Bennett, 1986) and to a problem-solving system that performs no learning. Two different training strategies are analyzed. The results demonstrate the efficacy of generalizing to *N*.

5.1. Experimental Methodology

Experiments are run using blocks-world inference rules. An initial situation is created by generating ten blocks, each with a randomly-chosen width and height. One at a time, they are dropped from an arbitrary horizontal position over a table; if they fall in an unstable location,

they are picked up and re-released over a new location. Once the ten blocks are placed, a randomly-chosen goal height is selected, centered above a second table. The goal height is determined by adding from one to four average block heights. In addition, the goal specifies a maximum height on towers. The difference between the minimum and maximum acceptable tower heights is equal to the maximum possible height of a block. This reason for this upper bound is explained later. A sample problem situation can be seen in figure 13.

Once a scene is constructed, three different problem solvers attempt to satisfy the goal. The first is called **no-learn**, as it acquires no new rules during problem solving. The second, called **sEBL**, is an implementation of a standard explanation-based generalization algorithm. (Explanation structures are pruned at terms that are either situation-independent or describe the initial state.) **BAGGER** is the third system. All three of these systems use a backward-chaining problem solver to satisfy the preconditions of rules. When the two learning systems attack a new problem, they first try to apply the rules they have acquired, possibly also using existing intra-situational rules. No inter-situational rules are used in combination with acquired rules, in order to limit searching, which would quickly become intractable. Hence, to be successful, an acquired rule must directly lead to a solution without using other inter-situational rules.

BAGGER's problem solver, in order to construct the *RIS*, is a slightly extended version of the standard backward-chaining problem solver used by the other two systems. First, the constraints on $?v_1$ are checked against the initial state. This leads to the binding of other components of the first vector in the sequence. Next, the problem solver checks if the last vector in the sequence (at this point, $?v_1$) satisfies the preconditions for $?v_n$. If so, a satisfactory sequence has been found and back-chaining terminates successfully. Otherwise, the last vector in the sequence is viewed as $?v_{i-1}$ and the problem solver attempts to satisfy the intermediate antecedent. This may lead to vector $?v_i$ being incorporated into the sequence. If a new vector is added, the final constraints on the sequence are checked again. If they are not satisfied, the new head of the sequence is viewed as

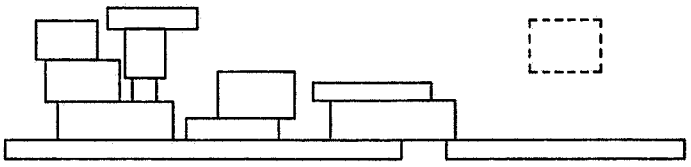

Figure 13. A Sample Problem

$?v_{i-1}$ and the process repeats. This cycle continues until either the current sequence satisfies the rule's antecedents or the initial state cannot support the insertion of another vector into the sequence. When the current sequence cannot be further extended, chronological back-tracking is performed, moving back to the last point where there are choices as how to lengthen the sequence.

Two different strategies for training the learning systems are employed. In one, called *autonomous mode*, the learning systems resort to solving a problem from "first principles" when none of their acquired rules can solve it. This means that the original inter-situational rules can be used, but learned rules are not used. When the proof of the solution to a problem is constructed in this manner, the systems apply their generalization algorithm and store any general rule that is produced. In the other strategy, called *training mode*, some number of solved problems (the *training set*) are initially presented to the systems, and the rules acquired from generalizing these solutions are applied to additional problems (the *test set*). Under this second strategy, if none of a system's acquired rules solves the problem at hand, the system is considered to have failed. No problem solving from first principles is ever performed by the learning systems in this mode.

Unfortunately, constructing towers containing more than two blocks from first principles exceeds the limits of the computers used in the experiments (Xerox Dandelions). For this reason, the performance of the **no-learn** system is estimated by fitting an exponential curve to the data obtained from constructing towers of size one and two. This curve is used by all three systems to estimate the time needed to construct towers from first principles when required, and a specialized procedure is used to generate a solution.

Data collection in these experiments is accomplished as follows. Initially, the two learning systems possess no acquired rules. They are then exposed to a number of sample situations, building up their rule collections according to the learning strategy applied. (At each point, all three systems address the same randomly-generated problem.) Statistics are collected as the systems solve problems and learn. This continues for a fixed number of problems, constituting an experimental run. However, a single run can be greatly effected by the ordering of the sample problems. To provide better estimates of performance, multiple experimental runs are performed. At the start of each run, the rules acquired in the previous run are discarded. When completed, the results of all the runs are averaged together. Unless otherwise noted, the data presented in this section is the result of superimposing 25 experimental runs and averaging. In all of the curves, solid circles represent data from **BAGGER**, open circles from **sEBL**, and x's from **no-learn**.

Each learning system stores its acquired rules in a linear list. During problem solving, these rules are tried in order. When a rule is successful, it is moved to the front of the list. This way, less useful rules will migrate toward the back of the list. Analysis of other indexing strategies is presented in (Shavlik, 1988), where a more comprehensive experimental analysis of EBL is presented.

This indexing strategy is the reason that, in the goal, tower heights are limited. The **sEBL** system would sooner or later encounter a goal requiring four blocks, and a rule for this would migrate to the front of its rule list. From that time on, regardless of the goal height, a four-block tower would be constructed. With a limit on tower heights, the rules for more efficiently building towers of lower heights have an opportunity to be tried. This issue would be exacerbated if the goal was not limited to four-block towers due to simulation time restrictions.

5.2. Experimental Results

In this section the operation of the two basic modes of operation — autonomous and training — are analyzed and compared. The autonomous mode is considered first. In this mode, whenever a system's current collection of acquired rules fails to solve a problem, a solution from first principles is constructed and generalized. Figure 14 shows the probability that the learning systems will need to resort to first principles as a function of the number of sample problems experienced. As more problems are experienced, this probability decreases. (On the first problem the probability is always 1.) **BAGGER** is less likely to need to resort to first principles than is **sEBL** because **BAGGER** produces a more general rule by analyzing the solution to the first problem.

On average, **BAGGER** learns 1.72 sequential-rules in each experimental run, while **sEBL** learns 4.28 rules. It takes **BAGGER** about 50 seconds and **sEBL** about 45 seconds to generalize a specific problem's solution. Averaging over problems 26—50 in each run (to estimate the asymptotic behavior), produces a mean solution time of 3720 seconds for **BAGGER**, 8100 seconds for **sEBL**, and 79,300 seconds[4] for **no-learn**. For **BAGGER**, this is a speed-up of 2.2 over **sEBL** and 21.3 over **no-learn**, where speed-up is defined as follows:

$$\textit{Speed-up of } A \textit{ over } B = \frac{\textit{mean solution time for } B}{\textit{mean solution time for } A}.$$

[4] One day contains 86,400 seconds.

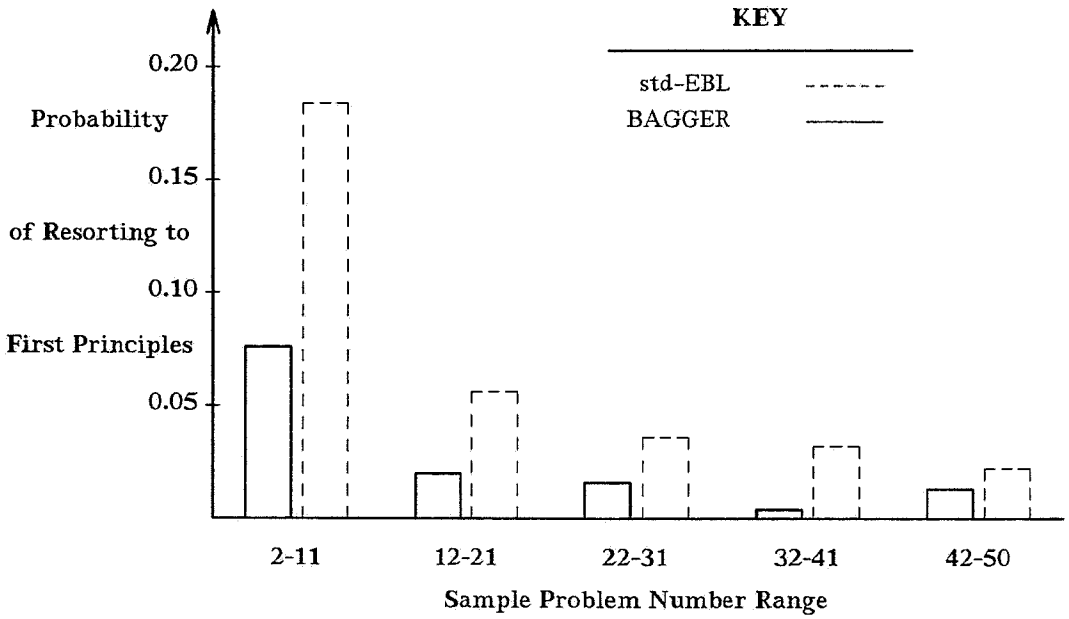

Figure 14. Probability of Resorting to First Principles in Autonomous Mode

Table 3 compares the speed of the three problem solvers over 625 sample problems (25 sample runs times the last 25 problems of each run). Recall that in each run, the three problem solvers all address the same problem at each point. The relative speeds of each are recorded and the table reflects how many times each system is the fastest, second fastest, and the slowest. Hence, **no-learn** solves about 20% of the problems faster than the two learning systems and about 60% of the problems slower than the other two. **BAGGER** solves slightly more than half the problems faster than do the other two systems. Only comparing the two learning systems, **BAGGER** solves about 70% of the problems faster than **sEBL** does.

Table 3 Relative Speed Summary in Autonomous Mode

	1st	2nd	3rd
No-Learn	20.2%	22.9	57.0
Std-EBL	24.8	41.6	33.6
BAGGER	55.0	35.5	9.4

BAGGER beats standard EBL: 71.8%

The numbers in this table only record the order of the three systems, they do not reflect by how much one system beats another. For example, building towers containing one block is often slightly faster to do from first principles, however towers of multiple blocks can be constructed much more rapidly by the learning systems. It takes **no-learn** about 10 seconds to build a tower with one block and 5×10^5 seconds for a tower of four blocks. For **BAGGER**, these averages are about 20 seconds and 70 seconds, respectively, for problems solved by its acquired rules. The performance separation seen in figure 14 is due to the fact that, when averaging numbers that vary by several orders of magnitude, the largest numbers heavily dominate.

Figure 15 plots the number of rules acquired as a function of problems experienced. The fact that the slopes of these curves are continually decreasing indicates that the time between learning episodes lengthens, which corresponds to the results of figure 14. That is, the mean time between failures of the acquired rules grows as more problems are experienced.

Figure 16 presents the performance during a single experimental run of the two learning systems in the autonomous mode. The average time to solve a problem is plotted, on a logarithmic scale, against the number of sample problems experienced. Notice that the time taken to produce a solution from first principles dominates the time taken to apply the acquired rules, accounting for the peaks in the curves.

Because the cost of solving a big problem from first principles greatly dominates the cost of applying acquired rules, the autonomous mode may not be an acceptable strategy. Although

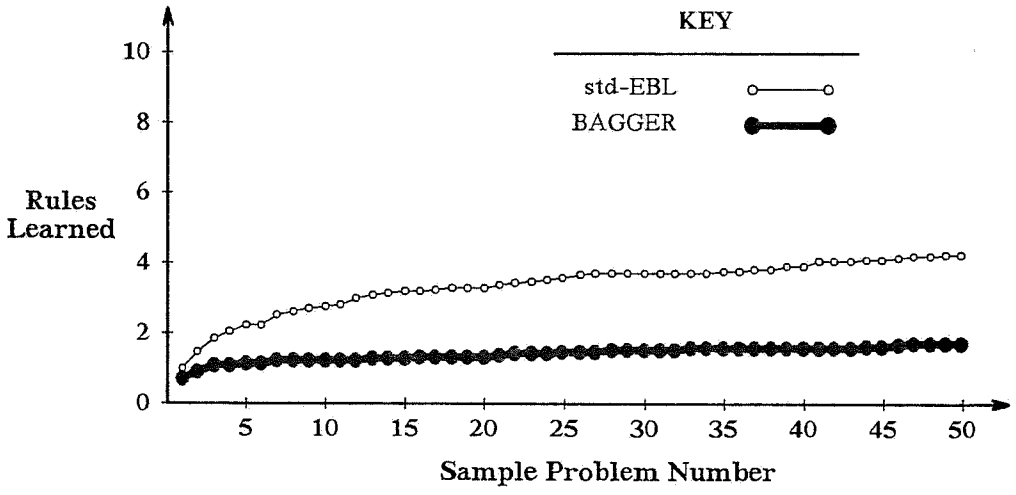

Figure 15. Rule Acquisition Comparison of the Autonomous Problem Solvers

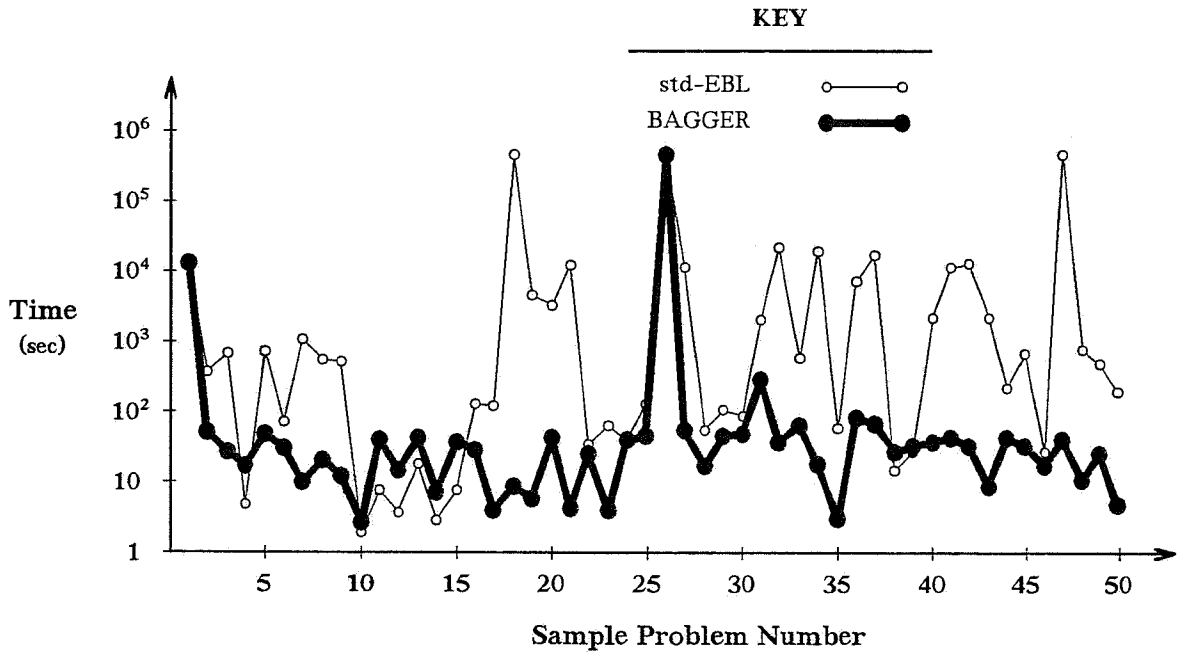

Figure 16. Performance Comparison of the Autonomous Problem Solvers

learning in this mode means many problems will be solved quicker than without learning, the time occasionally taken to construct a solution when a system's acquired rules fail can dominate the performance. The peaks in the right-side of figure 16 illustrate this. A long period may be required before a learning system acquires enough rules to cover all future problem-solving episodes without resorting to first principles.

The second learning mode provides an alternative. If an expert is available to provide solutions to sample problems and an occasional failure to solve a problem is acceptable, this mode is attractive. Here, a number of sample solutions are provided and the learning systems generalize these solutions, discarding new rules that are variants[5] of others already acquired. After training, the systems use their acquired rules to solve new problems. No problem solving from first principles is performed when a solution cannot be found using a system's acquired rules.

---

[5] The algorithm for detecting variants determines if two rules exactly match, given some renaming of variables. This means, for instance, that $a \wedge b$ and $b \wedge a$ are *not* variants. Hence, semantically equivalent rules are not always considered variants. A more sophisticated variant algorithm would reduce the number of saved rules. However if the variant algorithm considered associativity and commutativity, it would be much less efficient (Benanav, Kapur, and Narendran, 1985).

The performance results in the training mode are shown in figure 17. After ten training problems, the systems solve 20 additional problems. In these 20 test problems, the two learning systems never resort to using first principles. **BAGGER** takes, on average, 36.6 seconds on the test problems (versus 3720 seconds in the autonomous mode), **sEBL** requires an average of 828 seconds (versus 8100 seconds), and **no-learn** averages 68,400 seconds (versus 79,300 seconds).

Since **no-learn** operates the same in the two modes, these statistics indicate the random draw of problems produced an easier set in the second experiment. The substantial savings for the two learning systems (99% for **BAGGER** and 90% for **sEBL**) are due to the fact that in this mode these systems spend no time generating solutions from first principles. In this experiment, **BAGGER** has a speed-up of 22.6 over **sEBL** (versus 2.2 in the other experiment) and 1870 over **no-learn** (versus 23).

The relative speeds of the three systems in the training mode appear in table 4. (Only statistics from problems where all three problem solvers are successful are used to compute this table. As described later, this is about 98% of the test problems.) These numbers are comparable with the corresponding table for the autonomous mode. The main difference is that **sEBL** performs

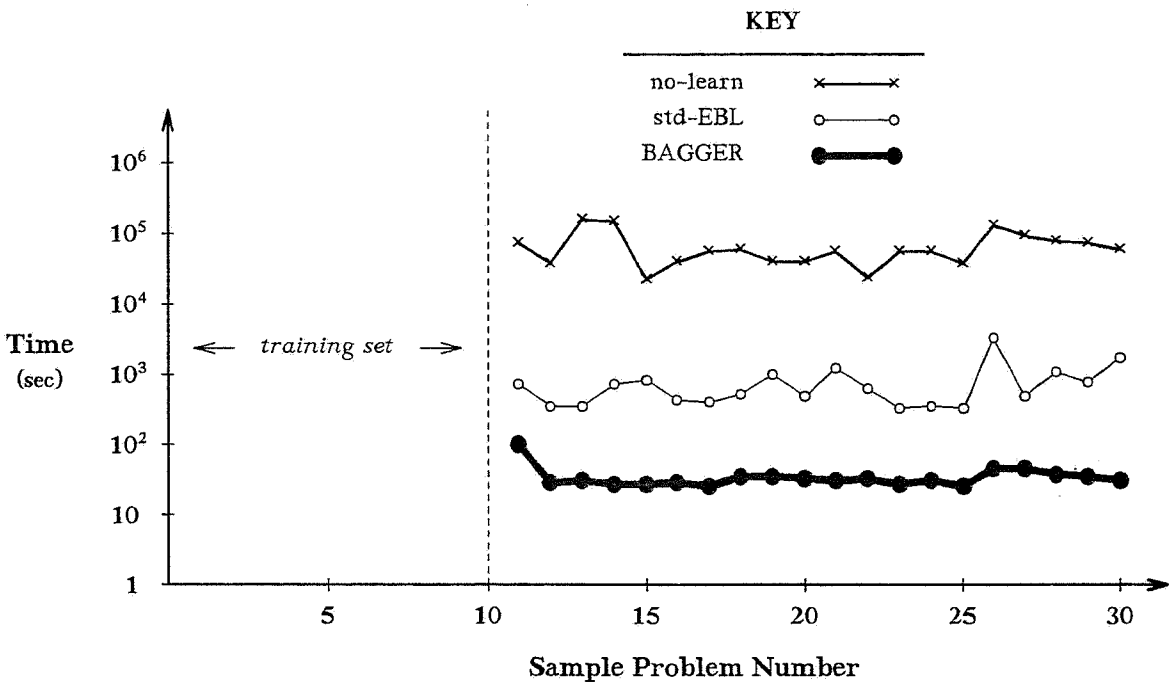

Figure 17. Performance Comparison of the Trained Problem Solvers

Table 4 Relative Speed Summary in Training Mode

	1st	2nd	3rd
No-Learn	20.8%	15.1	64.1
Std-EBL	21.6	47.8	30.7
BAGGER	57.6	37.1	5.2

BAGGER beats standard EBL: 75.4%

worse relative to the other systems (although its absolute performance is about ten times better than in the autonomous mode). The probable reason for this is that in the training mode the learning systems acquire more rules than in the autonomous mode.

The number of rules learned as a function of the size of the training set is plotted in figure 18. As before, **BAGGER** learns less rules than does **sEBL** and it approaches its asymptote sooner. Once the training set size exceeds about a half dozen examples, more rules are learned in the training mode than from 50 problems in the autonomous mode. This occurs because, during the training phase, new rules can be learned from problems that some previously-learned rule could have

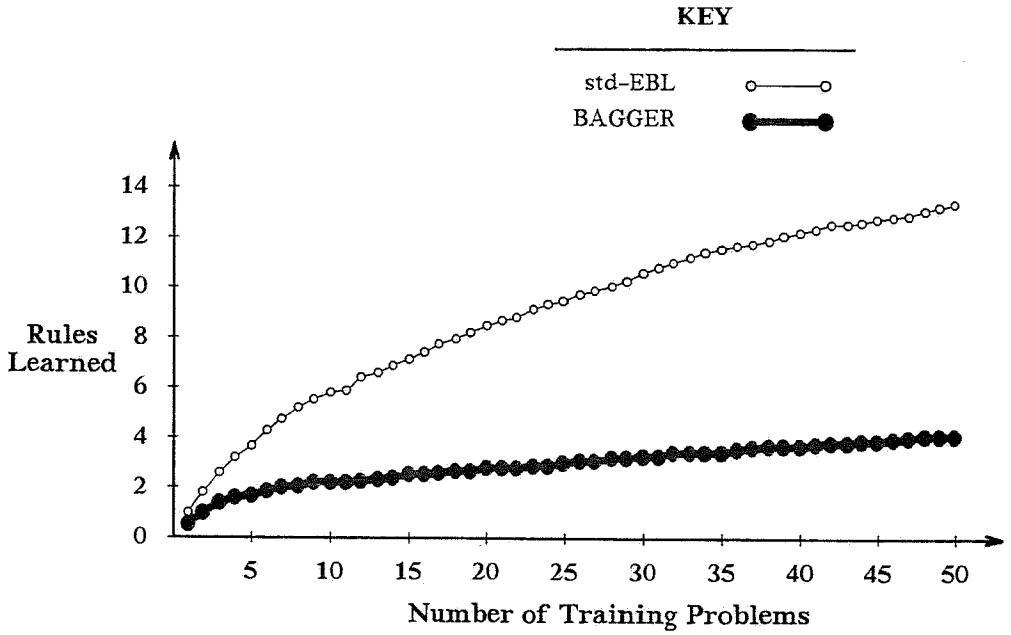

Figure 18. Rule Acquisition Comparison of the Trained Problem Solvers

solved, albeit in a different way than the expert's solution. Recall that in the training mode the expert's solution is immediately given — the systems do not first try to solve the problem. Only general rules that are simple syntactic variants of previously-acquired rules are discarded.

One of the costs of using the training mode is that occasionally the learning systems will not be able to solve a problem. Figure 19 plots the number of failures as a function of the size of the training set. In each experimental run used to construct this figure, 20 test problems are solved after the training examples are presented. With ten training solutions, both of the systems solve over 98.5% of the test problems.

The final figure in this section, figure 20, summarizes the performance of the three systems in the two training modes. Note that a logarithmic scale is used. Both of the experiments demonstrate the value of explanation-based learning and also show the advantages of the **BAGGER** system over standard explanation-based generalization algorithms. **BAGGER** solves most problems faster than do the other two systems, its overall performance is better, and it learns less rules than does **sEBL**. Comparing the two training modes demonstrates the value of external guidance to learning systems. If a system solves all of its problems on its own, the cost of occasionally solving complicated problems from first principles can dissipate much of the gains from learning. The remainder of this chapter investigates variants on these experiments, comparing the results to the data reported in this section.

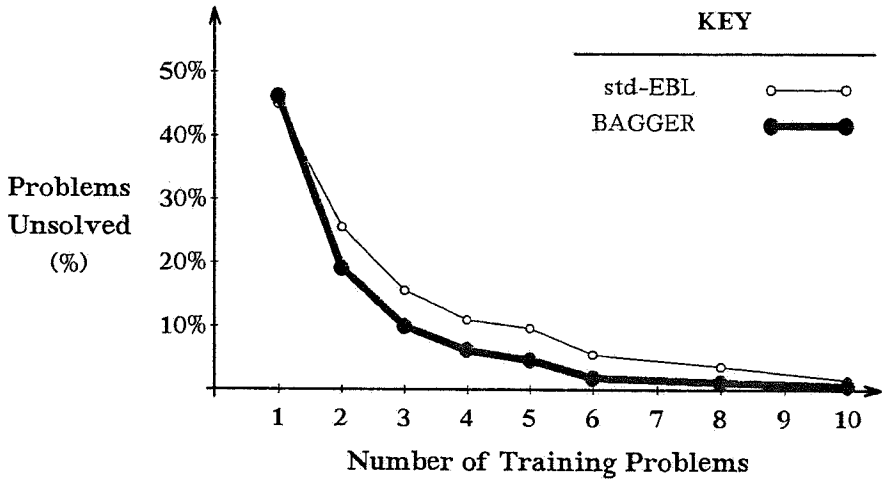

Figure 19. Failure Comparison of the Trained Problem Solvers

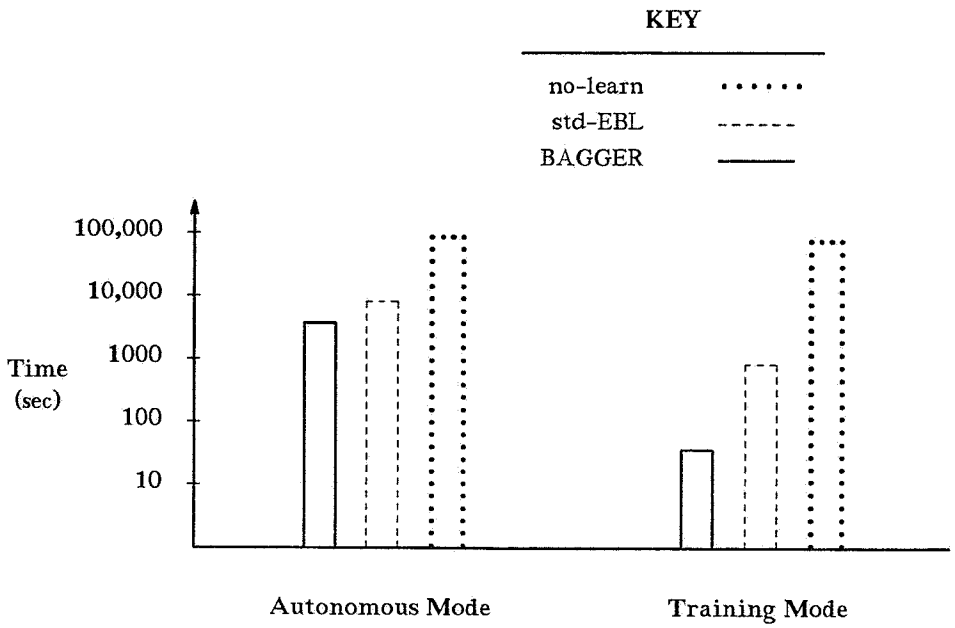

Figure 20. Performance of the Three Systems in the Two Modes

5.3. Summary

The empirical results presented in this section demonstrate the value of generalizing to N. In the two training strategies investigated, **BAGGER** performs substantially better than a system that performs no learning. **BAGGER** also outperforms a standard explanation-based learning system.

Other researchers have also reported on the performance improvement of standard EBL systems over problem solvers that do not learn (Fikes, Hart, and Nilsson, 1972; Minton, 1985; Mooney, 1988; O'Rorke, 1987b; Prieditis and Mostow, 1987; Steier, 1987). One major issue is that, as more new concepts are learned, problem-solving performance can *decrease*. This occurs because substantial time can be spent trying to apply rules that appear promising, but ultimately fail (Minton, 1985; Mooney, 1988). Also, a new broadly-applicable rule, which can require substantial time to instantiate, may block access to a more restricted, yet often sufficient, rule whose preconditions are easier to apply (Shavlik, DeJong, and Ross, 1987). While the non-learning system outperforms the learning systems on some problems, in the experiments reported in this section the overall effect is that learning is beneficial.

It may seem that investigating only tower-building problems unfairly favors explanation-based learning. An alternative would be to investigate a more diverse collection of problem types.

However, the negative effects of learning are manifested most strongly when the acquired concepts are closely related. If the effects of some rule support the satisfaction of a goal, substantial time can be spent trying to satisfy the preconditions of the rule. If this cannot be done, much time is wasted. To the **sEBL** system, a rule for stacking two blocks is quite different from one that moves four blocks. Frequently a rule that appears relevant fails. For example, often **sEBL** tries to satisfy a rule that specifies moving four blocks to meet the goal of having a block at a given height, only to fail after much effort because all combinations of four blocks exceed the limitations on the tower height. When the effects of a rule are clearly unrelated to the current goal, much less time is wasted, especially if a sophisticated data structure is used to organize rules according to the goals they support.

The **BAGGER** algorithm leads to the acquisition of fewer rules, because one of its rules may subsume many related rules learned using standard EBL. In this section's experiments, this decreases the likelihood that time will be wasted on rules that appear to be applicable. The probability that, in the training mode, a retrieved rule successfully solves a problem is 0.595 for **sEBL** and 0.998 for **BAGGER**. Additionally, fewer training examples are needed for **BAGGER** to acquire a sufficient set of new rules. These advantages over standard EBL magnify if the range of possible tower heights is increased (Shavlik, 1988).

The two training modes demonstrate the importance of external guidance to learning systems. In the autonomous mode, the systems must solve all problems on their own. The high cost of doing this when no learned rule applies dissipates much of the gains from learning. Substantial gains can be achieved by initially providing solutions to a collection of sample problems, and having the learners acquire their rules by generalizing these solutions. The usefulness of this depends on how representative the training samples are of future problems and how acceptable are occasional failures. Again, since **BAGGER** requires less training examples and produces more general rules, it addresses these issues better than does standard EBL.

6. RELATED WORK

The need to generalize number in EBL was first pointed out in (Shavlik and DeJong, 1985), where the knowledge that momentum is conserved for any N objects is learned from an example involving the collision of a fixed number of balls. Besides **BAGGER**, several other explanation-based approaches to generalizing number have been recently proposed.

Prieditis (1986) has developed a system which learns macro-operators representing sequences of repeated **STRIPS**-like operators. While **BAGGER** is very much in the spirit of Prieditis' work, **STRIPS**-like operators impose unwarranted restrictions. For instance, **BAGGER**'s use of predicate calculus allows generalization of repeated structure and repeated actions in a uniform manner. In addition, the **BAGGER** approach accommodates the use of additional inference rules to reason about what is true in a state. Everything does not have to appear explicitly in the focus rule. For example, in the stacking problem, other rules are used to determine the height of a tower and that an object is clear when the only object it supports is transferred. Also, instantiations of the focus rule do not have to directly connect — intervening inference rules can be involved when determining that the results of one instantiation partially support the preconditions of another. Prieditis' approach only analyzes the constraints imposed by the connections of the precondition, add, and delete lists of the operators of interest. There is nothing that corresponds to **BAGGER**'s unwinding operation nor are disjunctive rules learned.

In the **FERMI** system (Cheng and Carbonell, 1986), cyclic patterns are recognized using empirical methods and the detected repeated pattern is generalized using explanation-based learning techniques. A major strength of the **FERMI** system is the incorporation of conditionals within the learned macro-operator. However, unlike the techniques implemented in **BAGGER**, the rules acquired by **FERMI** are not fully based on an explanation-based analysis of an example, and so are not guaranteed to always work. For example, **FERMI** learns a strategy for solving a set of linear algebraic equations. None of the preconditions of the strategy check that the equations are linearly independent. The learned strategy will appear applicable to the problem of determining x and y from the equations $3x + y = 5$ and $6x + 2y = 10$. After a significant amount of work, the strategy will terminate unsuccessfully.

Cohen (1987) has recently developed and formalized another approach to the problem of generalizing number. His system generalizes number by constructing a finite-state control mechanism that deterministically directs the construction of proofs similar to the one used to justify the specific example. One significant property of his method is that it can generate proof procedures involving tree traversals and nested loops. A major difference between Cohen's method and other explanation-based algorithms is that in his approach no "internal nodes" of the explanation are eliminated during generalization. In other explanation-based algorithms, only the leaves of the operationalized explanation appear in the acquired rule. The generalization process guarantees that all of the inference rules within the explanation apply in the general case, and the

final result can be viewed as a compilation of the effect of combining these rules as generally as possible. Hence, to apply the new rule, only the general versions of these leave nodes need be satisfied. In Cohen's approach, every inference rule used in the original explanation is explicitly incorporated into the final result. Each rule may again be applied when satisfying the acquired rule. Hence, there is nothing in this approach corresponding to unwinding a rule from an arbitrary state back to the initial state, and the efficiency gains obtained by doing this are not achieved. Finally, because the final automaton is deterministic, it incorporates disjunctions only in a limited way. For example, if at some point two choices are equally general, the ordering in the final rule will be the same as that seen in the specific example.

A fourth system, **Physics 101** (Shavlik and DeJong, 1985, 1987a; Shavlik, 1988), differs from the above approaches in that the need for generalizing number is motivated by an analytic justification of an example's solution and general domain knowledge. This system learns such concepts as the general law of conservation of momentum (which is applicable to an arbitrary collection of objects) by observing and analyzing the solution to a specific three-body collision. In the momentum problem, information about number, localized in a single physics formula, leads to a global restructuring of a specific solution's explanation. However, **Physics 101** is designed to reason about the use of mathematical formulae. Its generalization algorithm takes great advantage of the properties of algebraic cancellation (e.g., $x - x = 0$). To be a broad solution of the generalization to N problem, non mathematically-based domains must also be handled.

Another aspect of generalizing the structure of explanations involves generalizing the *organization* of the nodes in the explanation, rather than generalizing the *number* of nodes. An approach of this form is presented in (Mooney, 1988), where the temporal order of actions is generalized in plan-based explanations. The approach is limited to domains expressed in the **STRIPS**-formalism (Fikes and Nilsson, 1971).

The problem of generalizing to N has also been addressed within the paradigms of similarity-based learning (Andreae, 1984; Dietterich and Michalski, 1983; Dufay and Latombe, 1984; Holder, *in preparation*; Michalski, 1983; Whitehall, 1987; Wolff, 1982) and automatic programming (Biermann, 1978; Kodratoff, 1979; Summers, 1977; Siklossy and Sykes, 1975). A general specification of number generalization has been advanced by Michalski (1983). He proposes a set of generalization rules including a *closing interval rule* and several *counting arguments rules* which can generate number-generalized structures. The difference between such similarity-based approaches

and **BAGGER**'s explanation-based approach is that the newly formed similarity-based concepts typically require verification from corroborating examples, whereas the explanation-based concepts are immediately supported by the domain theory.

7. SOME OPEN RESEARCH ISSUES

The **BAGGER** system has taken important steps towards the solution to the "generalization to N" problem. However, the research is still incomplete. From the vantage point of the current results, several avenues of future research are apparent.

One issue in generalizing the structure of explanations is that of deciding when there is enough information in the specific explanation to usefully generalize its structure. Due to the finiteness of a specific problem, fortuitous circumstances in the specific situation may have allowed shortcuts in the solution. Complications inherent in the general case may not have been faced. Hence the specific example provides no guidance as to how they should be addressed. In **BAGGER**, the requirement that, for an application of a focus rule to be generalized, it be viewable as the arbitrary *ith* application addresses the problem of recognizing fortuitous circumstances. If there is not enough information to view it as the *ith* application, it is likely that some important issue is not addressed in this focus rule application. However, more powerful techniques for recognizing fortuitous circumstances need to be developed.

Related to this, **BAGGER**'s method of choosing a focus rule needs improvement. Currently the first detected instance of interconnected applications of a rule is used as the focus rule. However, there could be several occurrences that satisfy these requirements. Techniques for comparing alternative focus rules are needed. Inductive inference approaches to detecting repeated structures (Andreae, 1984; Dietterich and Michalski, 1983; Dufay and Latombe, 1984; Holder, *in preparation*; Weld, 1986; Whitehall, 1987; Wolff, 1982) may be applicable to the generation of candidate focus rules, from which the explanation-based capabilities of **BAGGER** can build.

A second research topic is performing multiple generalizations to N in a single problem. Especially interesting is *interleaved generalization to N*. Here, in the final result, each application in an arbitrary length sequence would be supported by another sequence of arbitrary length. In other words, a portion of the intermediate antecedent of a **BAGGER** rule would be the antecedents of another **BAGGER** rule. Learning an interleaved sequential rule from one example may be too ambitious. A more reasonable approach may be to first learn a simple sequential rule, and then use

it in the explanation of a later problem. Managing the interactions between the two *RIS*'s is a major issue. See Cohen (1987) for a promising approach to the problem of interleaved generalization to *N*.

A third area of future research is to investigate how **BAGGER** and other such systems might acquire accessory inter-situational rules, such as frame axioms, to complement their composite rules. Currently, each of **BAGGER**'s new sequential inference rules specifies how to achieve a goal involving some arbitrary aggregation of objects by applying some number of operators. These rules are useful in directly achieving goals that match the consequent, but do not effectively improve **BAGGER**'s back-chaining problem-solving ability. This is because currently **BAGGER** does not construct new frame axioms for the rules it learns. (This problem is not specific to generalizing to *N*. Standard EBL algorithms must also face it when dealing with situation calculus.)

There are several methods of acquiring accessory rules. They can be constructed directly by combining the accessory rules of operators that make up the sequential rule. This may be intractable as the number of accessory rules for initial operators may be large and they may increase combinatorially in sequential rules. Another, potentially more attractive, approach is to treat the domain theory, augmented by sequential rules, as intractable. Since the accessory rules for learned rules are derivable from existing knowledge of initial operators, the approach in (Chien, 1987) might be used to acquire the unstated but derivable accessory rules when they are needed.

Investigating the generalization of operator application orderings within learned rules is a fourth opportunity for future research. Currently, in the rules learned by the **BAGGER** algorithm, the order interdependence among rule applications is specified in terms of sequences of vectors. However, this is unnecessarily constraining. When valid, these constraints should be specified in terms of *sets* or *bags*[6] of vectors. This could be accomplished by reasoning about the semantics of the system's predicate calculus functions and predicates. Properties such as symmetry, transitivity, and reflexivity may help determine constraints on order independence.

If a set satisfies a learned rule's antecedents, then *any* sequence derived from that set suffices. Conversely, if the vectors in a set fail to satisfy a rule's antecedents, there is no need to test each permutation of the elements. Unfortunately, testing all permutations occurs if the antecedents are unnecessarily expressed in terms of sequences. For example, assume the task at hand is to find

[6] A bag (or multi-set) is an *unordered* collection of elements in which an element can appear more than once.

enough heavy rocks in a storehouse to serve as ballast for a ship. A sequential rule may first add the weights in some order, find out that the sum weight of all the rocks in the room is insufficient, and then try another ordering for adding the weights. A rule specified in terms of sets would terminate after adding the weights once.

A fifth area of future research involves investigating the most efficient ordering of conjunctive goals. Consider an acquired sequential rule which builds towers of a desired height, subject to the constraint that no block can be placed upon a narrower block. The goal of building such towers is conjunctive: the correct height must be achieved and the width of the stacked blocks must be monotonically non-increasing. The optimal ordering is to select the blocks subject only to the height requirement and then sort them by size to determine their position in the tower. The reason this works is that a non-increasing ordering of widths on any set of blocks is guaranteed so that no additional block-selection constraints are imposed by this conjunct. The system should ultimately detect and exploit this kind of decomposability to improve the efficiency of the new rules.

Satisfying global constraints poses a sixth research problem. The sequential rules investigated in this chapter are all *incremental* in that successive operator applications converge toward the goal achievement. This is not necessarily the case for all sequential rules. Consider a sequential rule for unstacking complex block structures subject to the global constraint that the partially dismantled structure always be stable. Removing one block can drastically alter the significance of another block with respect to the structure's stability. For some structures, only the subterfuge of adding a temporary support block or a counter-balance will allow unstacking to proceed. A block may be safe to remove at one point but be essential to the over-all structural stability at the next, even though the block actually removed was physically distant from it. Such *non-incremental* effects are difficult to capture in sequential rules without permitting intermediate problem-solving within the rule execution.

The *RIS*, besides recording the focus rule's variable bindings, is used to store intermediate calculations, such as the height of the tower currently planned. Satisfying global constraints may require that the information in an *RIS* vector increase as the sequence lengthens. For example, assume that each block to be added to a tower can only support some block-dependent weight. The *RIS* may have to record the projected weight on each block while **BAGGER** plans the construction of a tower. Hence, as the sequence lengthens, each successive vector in the *RIS* will have to record information for one additional block. Figuratively speaking, the *RIS* will be getting longer and

wider. The current **BAGGER** algorithm does not support this.

Often a repeated process has a closed form solution. For example, summing the first N integers produces $\frac{N\,(N+1)}{2}$. There is no need to compute the intermediate partial summations. A *recurrence relation* is a recursive method for computing a sequence of numbers. Recognizing and solving recurrence relations during generalization is a seventh area for additional research.

Many recurrences can be solved to produce efficient ways to determine the *nth* result in a sequence. It is this property that motivates the requirement that **BAGGER**'s preconditions be expressed solely in terms of the initial state. However, the rule instantiation sequence still holds intermediate results. While often this information is needed (if, for instance, the resulting sequence of actions is to be executed in the external world), **BAGGER** would be more efficient if it could produce, whenever possible, number-generalized rules that did not require the construction of an *RIS*. If **BAGGER** observes the summation of, say, four numbers it will not produce the efficient result mentioned above. Instead it will produce a rule that stores the intermediate summations in the *RIS*. One extension that could be attempted is to create a library of templates for soluble recurrences, matching them against explanations. A more direct approach would be more appealing. Weld's (1986) technique of *aggregation* may be a fruitful approach. Aggregation is an abstraction technique for creating a continuous description from a series of discrete events.

The issue of termination is an eighth research area. One important aspect of generalizing number is that the acquired rules may produce data structures whose size can grow without bound (for example, the rule instantiation sequence in **BAGGER**) or the algorithms that satisfy these rules may fall into infinite loops (Cohen, 1987). Although the *halting problem* is undecidable in general, in restricted circumstances termination can be proved (Manna, 1974). Techniques for proving termination need to be incorporated into systems that generalize number. A practical, but less appealing, solution is to place resource bounds on the algorithms that apply number-generalized rules (Cohen, 1987), potentially excluding successful applications.

Finally, it is important to investigate the generalization to N problem in the context of imperfect and intractable domain models (Mitchell, Keller, and Kedar-Cabelli, 1986; Rajamoney and DeJong, 1987). In any real-world domain, a computer system's model can only approximate reality. Furthermore, the complexity of problem solving prohibits any semblance of completeness. Thus far **BAGGER**'s sequential rules have relied on a correct domain model, and it has not addressed issues of intractability, other than the use of an outside expert to provide sample solutions when

the construction of solutions from first principles is intractable.

8. CONCLUSION

Most research in explanation-based learning involves relaxing constraints on the variables in an explanation, rather than generalizing the number of inference rules used. This article presents an approach to the task of generalizing the structure of explanations. The approach relies on a shift in representation which accommodates indefinite numbers of rule applications. Compared to the results of standard explanation-based algorithms, more general rules are acquired, and since less rules need to be learned, better problem-solving performance gains are achieved.

To illustrate the approach, a situation calculus example from the blocks world is analyzed. This leads to a plan in which the number of blocks to be placed in a tower is generalized to N. In this example, the system observes three blocks being stacked upon one another, in order to satisfy the goal of having a block located at a specified height. Initially, the system has rules specifying how to transfer a single block from one location to another, and how the horizontal and vertical position of a block can be determined after is is moved. By analyzing the explanation of how moving three blocks satisfies the desired goal, **BAGGER** learns a new rule that represents how an unconstrained number of block transfers can be performed in order to satisfy future related goals.

The fully-implemented **BAGGER** system analyzes explanation structures (in this case, predicate calculus proofs) and detects repeated, inter-dependent applications of rules. Once a rule on which to focus attention is found, the system determines how an *arbitrary* number of instantiations of this rule can be concatenated together. This indefinite-length collection of rules is conceptually merged into the explanation, replacing the specific-length collection of rules, and an extension of a standard explanation-based algorithm produces a new rule from the augmented explanation.

Rules produced by **BAGGER** have the important property that their preconditions are expressed in terms of the initial state - they do not depend on the situations produced by intermediate applications of the focus rule. This means that the results of multiple applications of the rule are determined by reasoning only about the current situation. There is no need to apply the rule successively, each time checking if the preconditions for the next application are satisfied.

The specific example guides the extension of the focus rule into a structure representing an arbitrary number of repeated applications. Information not contained in the focus rule, but

appearing in the example, is often incorporated into the extended rule. In particular, *unwindable* rules provide the guidance as to how preconditions of the *ith* application can be specified in terms of the current state.

A concept involving an arbitrary number of substructures may involve any number of substantially different problems. However, a specific solution will have necessarily only addressed a finite number of them. To generalize to N, a system must recognize all the problems that exist in the general concept and, by analyzing the specific solution, surmount them. If the specific solution does not provide enough information to circumvent all problems, generalization to N cannot occur because **BAGGER** is designed not to perform any problem-solving search during generalization. When a specific solution surmounts, in an extendible fashion, a sub-problem in different ways during different instantiations of the focus rule, disjunctions appear in the acquired rule.

An empirical analysis of the benefit of generalizing the structure of explanations has been performed. These experiments indicate a performance improvement of at least an order of magnitude over standard explanation-based algorithms and several orders of magnitude over a problem solver that does not learn.

Generalizing to N is an important property currently lacking in most explanation-based systems. This research contributes to the theory and practice of explanation-based learning by developing and testing methods for extending the structure of explanations during generalization. It brings this field of machine learning closer to its goal of being able to acquire the full concept inherent in the solution to a specific problem.

ACKNOWLEDGEMENTS

The authors wish to thanks the other members of the explanation-based learning group at Illinois. Interactions with Raymond Mooney, Shankar Rajamoney, Scott Bennett, Steve Chien, and Melinda Gervasio have stimulated many interesting ideas.

APPENDIX: THE INITIAL INFERENCE RULES

The inference rules used in the tower-building (stacking) example are presented in this appendix. Not all the rules in the system are presented. However, a complete collection of the rules can be found in (Shavlik, 1988). The first table contains those rules that describe *intra*-situation inferences, while *inter*-situation inferences appear in the second table. The first rule in the second table is the definition of the transfer action. This rule is the *focus rule* of the stacking example. (The construct {*?a* | *?b*} matches a list with head *?a* and tail *?b*. For example, if matched with {*x*,*y*,*z*}, *?a* is bound to *x* and *?b* to {*y*,*z*}.)

Table A.1. Intra-Situation Rules Used in the Stacking Example

Rule	Description
Clear(?x,?s) FlatTop(?x) → FreeSpace(?x,?s)	If an object is clear and has a flat top, space is available.
Clear(?x,?s) Block(?x) → Liftable(?x,?s)	A block is liftable if it is clear.
Box(?x) → FlatTop(?x)	Boxes have flat tops.
Table(?x) → FlatTop(?x)	Tables have flat tops.
Box(?x) → Block(?x)	Boxes are a type of block.
Supports(?x,ϕ,?s) → Clear(?x,?s)	An object is clear if it is supporting nothing.
?x ≠ ?y → ?y ≠ ?x	Inequality is reflexive.
?x ≠ ?y NotMember(?x,?bag) → NotMember(?x,{?y | ?bag})	If two objects are distinct, and the first is not in a collection of objects, then the first is not a member of the collection that results from adding the second object to the original collection.
NotMember(?x,ϕ)	Nothing is a member of the empty set.
Member(?x,{?x})	Everything is a member of the singleton set containing it.
RemoveFromBag(?x,{?x | ?bag},?bag)	Remove this object from a collection of objects, producing a new collection of objects.

Table A.2. Inter-Situation Rules Used in the Stacking Example

Rule	Description
AchievableState(?s) Liftable(?x,?s) FreeSpace(?y,?s) ?x ≠ ?y → AchievableState(Do(Transfer(?x,?y),?s)) Clear(?x,Do(Transfer(?x,?y),?s)) On(?x,?y,Do(Transfer(?x,?y),?s))	If the top of an object is clear in some achievable state and there is free space on another object, then the first object can be moved from its present location to the new location. However, an object cannot be moved onto itself. Moving creates a new state in which the moved object is still clear but (possibly) at a new location.
Xpos(?y,?xpos,?s) → Xpos(?x,?xpos,Do(Transfer(?x,?y),?s))	After a transfer, the object moved is centered (in the X-direction) on the object upon which it is placed.
?x ≠ ?y Height(?x,?hx) Ypos(?y,?ypos,?s) ?ypos2 = (?hx + ?ypos) → Ypos(?x,?ypos2,Do(Transfer(?x,?y),?s))	After a transfer, the Y-position of the object moved is determined by adding its height to the Y-position of the object upon which it is placed.
?u ≠ ?y Supports(?u,?items,?s) NotMember(?x,?items) → Supports(?u,?items,Do(Transfer(?x,?y),?s))	If an object neither supports the moved object before the transfer, nor is the new supporter, then the collection of objects it supports remains unchanged.
?u ≠ ?y Supports(?u,?items,?s) Member(?x,?items) RemoveFromBag(?x,?items,?new) → Supports(?u,?new,Do(Transfer(?x,?y),?s))	If an object is not the new support of the moved object, but supported it before the transfer, then the moved object must be removed from the collection of objects being supported.

References

Ahn, W., Mooney, R. J., Brewer, W. F., and DeJong, G. F., "Schema Acquisition from One Example: Psychological Evidence for Explanation-Based Learning," *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, pp. 50-57, Seattle, Wash., July 1987.

Andreae, P. M., "Justified Generalization: Acquiring Procedures from Example," Ph. D. Thesis, Department of Electrical Engineering and Computer Science, MIT, Cambridge, Mass., January 1984. (Also appears as Technical Report 834, MIT AI Laboratory).

Benanav, D., Kapur, D., and Narendran, P., "Complexity of Matching Problems," *Proceedings of the First International Conference on Rewriting Techniques and Applications*, Dijon, France, May 1985.

Cheng, P., and Carbonell, J. G., "The FERMI System: Inducing Iterative Macro-operators from Experience," *Proceedings of the National Conference on Artificial Intelligence*, pp. 490-495, Philadelphia, Penn., August 1986.

Chien, S. A., "Simplifications in Temporal Persistence: An Approach to the Intractable Domain Theory Problem in Explanation-Based Learning," M.S. Thesis, Department of Computer Science, University of Illinois, Urbana, Ill., September 1987. (Also appears as Technical Report UILU-ENG-87-2255, AI Research Group, Coordinated Science Laboratory.)

Cohen, W. W., "A Technique for Generalizing Number in Explanation-Based Learning," Technical Report ML-TR-19, Department of Computer Science, Rutgers University, New Brunswick, New Jersey, September 1987.

DeJong, G. F., and Mooney, R. J., "Explanation-Based Learning: An Alternative View," *Machine Learning*, Vol. 1, No. 2, pp. 145-176, April 1986.

Dietterich, T., and Michalski, R. S., "Discovering Patterns in Sequences of Objects," *Proceedings of the 1983 International Machine Learning Workshop*, Urbana, Ill., pp. 41-57, June 1983.

Dufay, B., and Latombe, J., "An Approach to Automatic Robot Programming Based on Inductive Learning," in *Robotics Research: The First International Symposium*, MIT Press, pp. 97-115, Cambridge, Mass., 1984.

Ellman, T., "Generalizing Logic Circuit Designs by Analyzing Proofs of Correctness," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pp. 643-646, Los Angeles, Calif., August 1985.

Ellman, T., "Explanation-Based Learning: A Survey of Programs and Perspectives," Technical Report, Department of Computer Science, Columbia University, New York City, New York, May 1987.

Fahlman, S., "A Planning System for Robot Construction Tasks," *Artificial Intelligence*, Vol. 5, No. 1, pp. 1-49, 1974.

Fikes, R. E., and Nilsson, N. J., "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence*, Vol. 2, No. 3/4, pp. 189-208, 1971.

Fikes, R. E., Hart, P. E., and Nilsson, N. J., "Learning and Executing Generalized Robot Plans," *Artificial Intelligence*, Vol. 3, No. 4, pp. 251-288, 1972.

Fikes, R., "Deductive Retrieval Mechanisms for State Description Models," *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, pp. 99-106, Tiblisi, Georgia, U.S.S.R., August 1975.

Green, C. C., "Application of Theorem Proving to Problem Solving," *Proceedings of the First International Joint Conference on Artificial Intelligence*, pp. 219-239, Washington, D.C., August 1969.

Hirsh, H., "Explanation-Based Generalization in a Logic-Programming Environment," *Proceedings of the Tenth*

International Joint Conference on Artificial Intelligence, pp. 221-227, Milan, Italy, August 1987.

Holder, L. B., "Discovering Substructures in Examples," M.S. Thesis (in preparation), Department of Computer Science, University of Illinois, Urbana, Ill.

Kedar-Cabelli, S. T., and McCarty, L. T., "Explanation-Based Generalization as Resolution Theorem Proving," *Proceedings of the 1987 International Machine Learning Workshop*, pp. 383-389, Irvine, Calif., June 1987.

Keller, R. M., "Defining Operationality for Explanation-Based Learning," *Proceedings of the National Conference on Artificial Intelligence*, pp. 482-487, Seattle, Wash., July 1987.

Kodratoff, Y., "A Class of Functions Synthesized from a Finite Number of Examples and a LISP Program Scheme," *International Journal of Computer and Information Sciences*, Vol. 8, No. 6, pp. 489-521, 1979.

Laird, J., Rosenbloom, P., and Newell, A., "Chunking in Soar: The Anatomy of a General Learning Mechanism," *Machine Learning*, Vol. 1, No. 1, pp. 11-46, January 1986.

Manna, Z., *Mathematical Theory of Computation*, McGraw-Hill, New York, NY, 1974.

McCarthy, J., "Situations, Actions, and Causal Laws," memorandum, Stanford University, Stanford, Calif., 1963. (Reprinted in *Semantic Information Processing*, M. Minsky (Ed.), MIT Press, pp. 410-417, Cambridge, Mass., 1968.)

Michalski, R. S., "A Theory and Methodology of Inductive Learning" in *Machine Learning: An AI Approach*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), pp. 83-134, Tioga, Palo Alto, Calif, 1983.

Minton, S. N., "Selectively Generalizing Plans for Problem-Solving," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pp. 596-599, Los Angeles, Calif., August 1985.

Minton, S. N., Carbonell, J. G., Etzioni, O., Knoblock, C. A., Kuokka, D. R., "Acquiring Effective Search Control Rules: Explanation-Based Learning in the PRODIGY System," *Proceedings of the Fourth International Workshop on Machine Learning* Artificial Intelligence, pp. 122-133, Irvine, Calif., June 1987.

Mitchell, T. M., Mahadevan, S., and Steinberg, L. I., "LEAP: A Learning Apprentice for VLSI Design," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pp. 573-580, Los Angeles, Calif., August 1985.

Mitchell, T. M., Keller, R., and Kedar-Cabelli, S., "Explanation-Based Generalization: A Unifying View," *Machine Learning*, Vol. 1, No. 1, pp. 47-80, January 1986.

Mooney, R. J., and Bennett, S. W., "A Domain Independent Explanation-Based Generalizer," *Proceedings of the National Conference on Artificial Intelligence*, pp. 551-555, Philadelphia, Penn., August 1986.

Mooney, R. J., "A General Explanation-Based Learning Mechanism and its Application to Narrative Understanding," Ph.D. Thesis, Department of Computer Science, University of Illinois, Urbana, Ill., 1988. (Also appears as Technical Report UILU-ENG-87-2269, AI Research Group, Coordinated Science Laboratory.)

Nilsson, N. J., *Principles of Artificial Intelligence*, Tioga, Palo Alto, Calif., 1980.

O'Rorke, P. V., "Explanation-Based Learning via Constraint Posting and Propagation," Ph.D. Thesis, Department of Computer Science, University of Illinois, Urbana, Ill., 1987a. (Also appears as Technical Report UILU-ENG-87-2239, AI Research Group, Coordinated Science Laboratory.)

-------, "LT Revisited: Experimental Results of Applying Explanation-Based Learning to the Logic of Principia Mathematica," *Proceedings of the 1987 International Machine Learning Workshop*, pp. 148-159, Irvine,

Calif., June 1987b.

Prieditis, A. E., "Discovery of Algorithms from Weak Methods," *Proceedings of the International Meeting on Advances in Learning*, pp. 37-52, Les Arcs, Switzerland, 1986. (An updated version appears in *Machine Learning: An Artificial Intelligence Approach*, Vol. III, R. S. Michalski and Y. Kodratroff (Eds.), Morgan Kaufman, Los Altos, Calif.)

Prieditis, A. E. and Mostow, J., "PROLEARN: Towards a Prolog Interpreter that Learns," *Proceedings of the National Conference on Artificial Intelligence*, pp. 494-498, Seattle, Wash, July 1987.

Rajamoney, S., and DeJong, G. F., "The Classification Detection and Handling of Imperfect Theory Problems," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pp. 205-207, Milan, Italy, August 1987.

Rendell, L., "Substantial Constructive Induction using Layered Information Compression: Tractable Feature Formation in Search," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pp. 650-658, Los Angeles, Calif., August 1985.

Segre, A. M., "On the Operationality/Generality Trade-off in Explanation-Based Learning," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pp. 242-248, Milan, Italy, August 1987.

Shavlik, J. W., "Generalizing the Structure of Explanations in Explanation-Based Learning," Ph.D. Thesis, Department of Computer Science, University of Illinois, Urbana, Ill., 1988. (Available as a technical report from AI Research Group, Coordinated Science Laboratory.)

Shavlik, J. W., and DeJong, G. F., "Building a Computer Model of Learning Classical Mechanics," *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, pp. 351-355, Irvine, Calif, August 1985.

-------, "Analyzing Variable Cancellations to Generalize Symbolic Mathematical Calculations," *Proceedings of the Third IEEE Conference on Artificial Intelligence Applications*, Orlando, Fla., February 1987a.

-------, "BAGGER: An EBL System that Extends and Generalizes Explanations," *Proceedings of the National Conference on Artificial Intelligence*, pp. 516-520, Seattle, Wash., July 1987b.

-------, "An Explanation-Based Approach to Generalizing Number," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pp. 236-238, Milan, Italy, August 1987c.

Shavlik, J. W., DeJong, G. F., and Ross, B. H., "Acquiring Special Case Schemata in Explanation-Based Learning," *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, pp. 851-860, Seattle, Wash., July 1987.

Siklossy, L., and Sykes, D. A., "Automatic Program Synthesis from Example Problems," *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, pp. 268-273, Tbilisi, Georgia, USSR 1975.

Steier, D., "CYPRESS-Soar: A Case Study in Search and Learning in Algorithm Design," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pp. 327-330, Milan, Italy, August 1987.

Summers, P. D., "A Methodology for LISP Program Construction from Examples," *Journal of the Association for Computing Machinery*, Vol. 24, pp. 161-175, 1977.

Waldinger, R., "Achieving Several Goals Simultaneously," in *Machine Intelligence 8*, E. Elcock and D. Michie (Eds.), Ellis Horwood Limited, London, 1977.

Weld, D. S., "The Use of Aggregation in Casual Simulation," *Artificial Intelligence*, Vol. 30, No. 1, pp. 1-34, October 1986.

Whitehall, B. L., "Substructure Discovery in Executed Action Sequences," M.S. Thesis, Department of Computer Science, University of Illinois, Urbana, Ill., September 1987. (Also appears as Technical Report UILU-ENG-87-2256, AI Research Group, Coordinated Science Laboratory).

Wolff, J. G., "Language Acquisition, Data Compression and Generalization," *Language and Communication*, Vol. 2, No. 1, pp. 57-89, 1982.