Verification-Based Learning: A Generalisation Strategy for Inferring Problem-Reduction Methods

Sridhar Mahadevan*

Computer Science Department
Rutgers University
New Brunswick, NJ 08903

Abstract

A major impediment to the development of high-performance knowledge-based systems arises from the prohibitive effort involved in equipping these systems with a sufficient set of problem-solving methods. Thus, one important research problem in Machine Learning has been the study of techniques for inferring problem-solving methods from examples. Although a number of techniques for learning problem-solving methods have been described in the literature, all of them assume a *state-space* model of problem-solving. In this paper we describe a new technique for learning problem-reduction methods, Verification-Based Learning (VBL), which extends the earlier techniques to the problem-reduction formulation of problem-solving. We illustrate the VBL technique with examples drawn from circuit design and symbolic integration.

I Introduction

A. Motivation

Knowledge-based systems require a large number of *domain-specific* problem-solving methods for achieving high levels of performance. The VEXED knowledge-based system for circuit design |1|, and the PECOS system for knowledge-based automatic programming |2] are a few examples of systems that need a large set of domain-specific problem-solving methods. Building knowledge-based problem-solvers has thus been a laborious process, because of the effort needed in equipping these systems with a sufficient set of problem-solving methods.

Earlier researchers in the field of Machine Learning have addressed the knowledge acquisition issue by developing a number of techniques for learning problem-solving methods from examples; however, all these techniques assume a *state-space* problem-solving model. For example, in the plan generalisation component of STRIPS [3|, a planning method is viewed as a mapping, from an initial state description into a terminal state description; new methods are constructed as *macros* of primitive planning methods. Work on learning *control* knowledge for selecting preferred methods, specifically (4, 6], has also adopted a state-space model: the knowledge specifying when a method *should* be applied is determined by computing the *weakest precondition* of a sequence, containing that particular method, which maps some initial state into a specific goal state (such as a *solved* problem, or a *won* state).

In order to cope with the complexity of certain design planning domain-. knowledge' Þa - p ,f^< <'lwr^ MI SUch domains, have frequently adopted a *problem reduction* approach to solving problems Thus we need to develop techniques for learning problem-solving methods which are appropriate to the problem-reduction formulation of problem-solving The primary contribution of this paper is to present a new technique for inferring general problem-reduction methods from training examples of decompositions of specific problems. This technique, Verification-Based Learning(VBL), can be viewed as an extension of earlier techniques, particularly that of |3, 6), to the problem-reduction formulation of problem-solving.

This research arose in the context of developing LEAP, a *Learning Apprentice* system for circuit design |7] By a Learning Apprentice system, we mean one that is meant to act as an *interactive* problem-solving aid, and is specifically designed to augment its knowledge base by monitoring and analyzing the problem-solving activity of its users.** In those situations, where it is unable to provide advice, or when its advice is rejected by the user, LEAP will augment its knowledge of circuit design, by analyzing and generalizing the solution provided by the user to form a new problem-reduction method. At present we have implemented a prototype version of the LEAP system. The example of a problem-reduction method in circuit design that we describe later in this paper is one of several instances of circuit decompositions we have used to test this prototype version.

B. The Problem-Reduction Formulation of Problem-Solving

We now give a more precise description of the model of problem-solving that we use in this paper. The problem-reduction formulation of problem-solving has been well-studied |8, 9].*** In this formulation, states describe *problem instances,* the initial state is the description of the problem being solved, the final state is a solution to the original problem, and a problem-reduction method, which is a mapping between states, is one that decomposes a given problem into a number of *simpler* subproblems, such that *the solution to the original problem it obtained by tome eompotition of the solutions to each of the subproblems.*

For the purposes of this paper, we define problem-reduction methods to be the following mapping,

$$P \rightarrow M \rightarrow C\{P_1, P_2, \ldots, P_N\}$$

•This material is bawd on work supported by the Defease Advanced Research Projects Agency under Research Contract N0O014-S1.K-0394. The views and conclusions contained in tab document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defease Advanced Research Projects Agency or the U.S. Government.

•Thus, gaining expertise by "looking over their shoulders"

***The distinction between state space methods and problem-reduction methods is alternatively described in [6] as tbat between *production-type* methods and r*eduction-type* metbods.

where P is the specification of a problem (in some language), which is decomposed by the method M into a set of subproblems $P_1 \ldots, P_N$. The *combinator* C describes how the solutions to each of the subproblems P_i are to be combined to yield the solution to the original problem. We now illustrate the abstract notion of a problem-reduction method, with the following example chosen from the domain of symbolic integration.

Integral-of-a-Sum Method :

$$\int \{f_1(x) + f_2(x)\}dx \rightarrow M \rightarrow \int f_1(x)dx + \int f_2(x)dx$$

The above method *decomposes* the problem of integrating a sum of two functions into a set of *subproblems,* that of integrating each of the summands. Using the terminology introduced earlier, the problem P is the integral $\int \{f_1(x) + f_2(x)\}dx$, the two subproblems P_1 and P_2 are $\int f_1(x)dx$ and $\int f_2(x)dx$ respectively, and the combinator C is addition.

C. Using Verification as a basis for Generalisation

We now introduce the idea of using *verification* as a basis for the generalization of problem-reduction methods from instances. We represent an instance of the application of a problem-reduction method as an ordered *pair* of states <x,y>; x represents an instance of the class of problems to which the method *can* be applied, and y represents the composition of some set of subproblems whose solution implies the solution to the original problem x Given such an instance of an *unknown* problem-reduction method as a pair of states <x,y>, the generalisation problem that we address in this paper consists in determining this general method from the instance. As a first step towards the generalization, the program verifies that the solution to the composition of the subproblems, y, *implies* the solution to the original problem x. The second (and final) step involves the central idea underlying the VBL technique, *the unknown method can be determined by generalizing the problem-states x and y, retaining only those features of these problem-states that were important for the purpose of verification.*

The idea of using verification as a basis for a learning technique, is related to similar schemes used in other *analytical* learning techniques, specifically *Goal-Directed Learning* |4], *Explanation-Based Learning* [10] and *Constraint-Based Generalization* (5). All these learning techniques are similar in that they first generate an explanation or proof (here, a verification) of why the given training instance (here, instances represent applications of problem-reduction methods) satisfies a particular goal (here, the goal is showing that the decomposition of the problem instance was a correct one), and then they generalise the instance (here, forming a new problem-reduction method) using the constructed explanation or proof to constrain the generalization.

D. Outline of the paper

- In section 2, we state the generalisation problem for inferring problem-reduction methods from examples.
- Sections 3 and 4 contain two detailed examples of the application of VBL to the task of inferring problem-reduction methods in circuit design and symbolic integration.
- Finally, in section 5, we summarise by viewing VBL from a number of different perspectives, outlining some of its limitations and describing some work in progress on them.

II The Generalisation Problem for Inferring Problem-Reduction Methods

A. Statement of the Problem

Before proceeding to give domain-specific examples of VBL, as we will be doing in sections S and 4, it is important that we state the generalisation problem that this paper addresses, in domain-independent terms. Figure II-1 provides such a statement.

Figure II-1: The (ieneralizaiion Problem for Inferring Problem-Reduction Methods

- Given -
  - o A language of *instances* of problems.
  - o A language of *generalizations* of problems. Each generalization in this language describes some class of problem instances.
  - o A single positive instance of a problem-reduction, which is composed of the following pair.
    1. Specification of a problem P in the instance language.
    2. Specification of a set of subproblems P. in the instance language and a combinator C.
  - o Some domain theory, in the form of a set of *transformations,* which can be used to verify assertions of the form -- the solution to the composition of the set of subproblems P. *implies* the solution to the problem P. More formally, such assertions can be stated as

$$C\{P_1, P_2, \ldots, P_N\} \Rightarrow P$$

- Determine -
  - o Description of a problem-reduction method in the generalisation language that is consistent with the observed instance; this is computed by generalising the specifications P and P., using the verification of the above assertion to constrain the generalisation.

B. Discussion

The domain theory required by VBL, in order to construct a proof or verification of the assertion in figure II-1, is a set of *transformations;* each transformation can be viewed as a primitive problem-reduction method, and new problem-reduction methods can be viewed as being obtained by composing these primitive methods together in a specific manner. Forming a new problem-solving method by composing together a sequence of primitive methods is a general strategy, which has been used as the basis for many earlier techniques [3, 111. In the next section, we will provide an example from circuit design, which illustrates how we use this approach to infer new problem-reduction methods.

III Learning Problem-Reduction Methods for Circuit Design

In this section we describe an application of VBL to the task of acquiring problem-reduction methods for circuit design. We begin by formulating circuit design as a problem-reduction process. We then focus on a particular problem-reduction method for designing a small class of circuit specifications, first describing the method itself, and subsequently, in a number of detailed steps, showing how it may be acquired from a single training instance.

A. Circuit Design as Problem Reduction

In order to view circuit design as a problem-reduction process, we need to specify the various; components that constitute the problem-reduction model. The language of instances of problems corresponds to the *functional* specifications of circuits; these define the mapping between the input and output signals of a circuit, (for example, the output of an adder equals the *turn* of its inputs.) The initial state is the functional specification of a circuit to be designed, the final state is an implementation of the circuit specification in terms of a set of primitive components, and a problem-reduction method is one that decomposes a given circuit specification P into specifications P_i of a number of interconnected *ttmpicr* circuits (submodules) The language of generalisations of problems enhances the expressive capabilities of the instance language, by including the ability to specify *arbitrary* boolean functions as part of functional specifications **** We thus view the problem of learning problem-reduction methods in circuit design, as an instance of the more general problem of inferring problem-reduction methods from examples.

B. An Dlustration of Problem-Reduction in Circuit Design

For the sake of concreteness in the discussion of the VBL technique, we need to consider a simple example of a problem-reduction method in circuit design, and phrase the remainder of the discussion in terms of this example. We first describe the method itself, and then provide an example of its use.

Figure III-I provides a simple example of a problem-reduction method that suggests one plausible way of implementing *a conjunction of any two boolean expressions* The left-hand side (LHS) of the problem-reduction method describes the class of specifications to which it *can* be applied. The right-hand side (RHS) of the method suggests both a decomposition of the specification in the LHS into specifications for a set of submodules, and also a way of interconnecting them.

A training example that represents an instance of the above method is given in figure III-2. In this example the circuit being designed is a *product-of-sums* circuit whose specification P is as given in the figure. The context here is that of a Learning Apprentice system, which being ignorant of the above general method for implementing a conjunction of two boolean expressions, may suggest using an AND gate and two OR gates as one way of implementing the circuit specification P. The user steps in at this point, and disregarding the system's suggestion, provides his preferred way of implementing the specification P, which is to use a set of NOR gates interconnected as shown in figure III-2.

""For reasons of clarity in tbt ensuing discussion of the learning method, we adopt a simplified representation of functional specifications of circuits. Specifically, this representation omits any reference to attributes of signals such as timing' and sncoding. [7] provides further details of this representation.

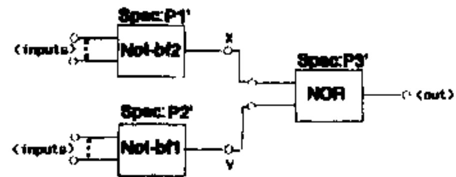
The Learning Apprentice views the user-supplied example as representing an instance of the application of an unknown problem-reduction method, and sets itself the task of inferring the general method. The general problem-reduction method that is inferred from the training instance in figure III-2, is the one we described earlier, in figure III-1. In the next few paragraphs, we provide a detailed description of the use of VBL in determining this general method.

Figure III-1: A Problem-Reduction Method for Circuit Design

LHS: IF the functional specification to be implemented is**

(BQUALS <out> (AND <bool-fn2> <bool-fn1>))

RHS: THEN one possible implementation is



with specifications of the three submodules as follows:

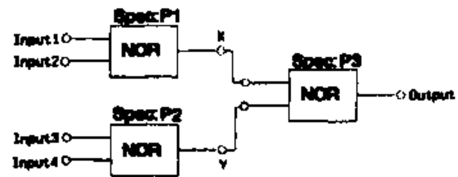
P_1': (EQUALS X (NOT <bool-fn2>))
P_2': (EQUALS Y (NOT <bool-fn1>))
P_3': (EQUALS <out> (NOT (OR X Y)))

Figure III-2: A Training Instance of the Problem-Reduction Method in figure III-I

Functional specification to be Implemented P:

(BQUALS Output (AND (OR $Input_1$ $Input_2$)
(OR $Input_3$ $Input_4$)))

User's Solution: Product-of-Sums circuit



where
P_1: (EQUALS X (NOT (OR $Input_1$ $Input_2$)))
P_2: (EQUALS Y (NOT (OR $Input_3$ $Input_4$)))
P_3: (EQUALS Output (NOT (OR X Y)))

C. Step 1: Forming the Composed Specification

The first step in the process of inferring a general problem-reduction method from a training example is verifying the correctness of the decomposition for the given example. In circuit design, this corresponds to ensuring that the function computed by the decomposed circuit meets the original circuit specification. Before attempting to construct a proof of correctness, we must have some way of determining the function computed by the decomposed circuit. It is to this matter that we turn to next, in our discussion of VBL.

*<bool-fn1> abort represents An arbitrary boolean function.

By composing the specifications of each of the submodules constituting a given circuit, in a way that depends on the interconnection of the submodules, we can [illegible] a specification of the circuits function from its structure This *composed specification,* can be computed by a simple substitution process, and is determined by the relationships between signals at various points in the circuit. For example, we can obtain a relationship between the output signal and the inputs of the product-of-sums circuit in figure II1-2, by substituting for X and Y in the specification P_3, their relationships to the input signals described in P_1 and P_2 Carrying this out, we obtain the following composed specification for the product-of-sums circuit

```
Output = (NOT (OR (NOT (OR Input1 Input2))
                  (NOT (OR Input3 Input4))))
```

Several remarks may now be made, in connection with our definition of a composed specification, which are of importance to the generalization process that follows.

- In general, the composed specification will be a *reexpresscd* version of the original specification. This is due to *constraints* on the possible structures that primitive circuits (problems) can take. For example, in VLSI design, since circuits naturally *invert* their inputs, specifications in terms of AND and OR boolean functions must often be reexpressed in order to obtain implementations using combinations of NAND and NOR gates. Thus, since we must show that the composed specification implies the original specification, this motivates the need for a verification of the correctness of a decomposition.
- The form of the composed specification depends on the structure of the decomposed circuit (problem). We will make use of this property, when generalizing each of the submodule (subproblem) specifications.

D. Step 2: The Process of Verification

Having obtained the composed specification of a decomposed circuit (problem), the next step in the VBL technique is to construct a proof of the correctness of the decomposition. That is, we need to verify the truth of the following assertion, which we repeat from figure II-1, and which states that the composed specification must imply the original specification.

$$C\{P_1, P_2, \ldots, P_N\} \Rightarrow P$$

For the product-of-sums circuit example, by substituting its composed and original specifications in the above expression, we obtain the following assertion, whose verification is the topic of this section.

```
(NOT (OR (NOT (OR Input1 Input2))
         (NOT (OR Input3 Input4))))
   =>
         (AND (OR Input1 Input2)
              (OR Input3 Input4))
```

If we are only concerned about verifying arbitrary assertions of the above form, it is clear that there exists considerable latitude in the choice of an appropriate scheme for verification. Many such schemes have been developed by researchers interested in *circuit verification [12].* However, since we view verification as only a means to our end of wanting to infer problem-reduction methods, any scheme we choose must meet the following additional requirement - *it should be possible to use the proof constructed by the verification scheme [illegible] those features of the specifications that were needed for the proof to carry through****

We now describe a verification scheme satisfying the above requirement We construct the proof as a sequence of *transformations,* which will yield the original specification when applied to the composed specification Each such transformation, which can be viewed as a primitive problem-reduction method, is specified by its *precondition* - the class of specifications to which it can be applied - and its *postcondition,* which describes the result of applying the transformation. Constructing a verification as a sequence of such transformations satisfies the above requirement, since it enables us to determine from the sequence a generalisation of the circuit specifications P and P_i, using *constraint propagation.* Two examples of transformations that we will use in the current example are given below.

De-Morgan's Law:
Precondition:
```
(NOT (OR <bool-fn1> <bool-fn2>) )
```
Postcondition:
```
(AND (NOT <bool-fn1>)
     (NOT <bool-fn2>) )
```

Remove-Double-Negation:
Precondition:
```
(NOT (NOT <bool-fn>) )
```
Postcondition:
```
<bool-fn>
```

Using the scheme mentioned above, we now provide, for the product-of-sums circuit example, a verification of the correctness of the user-suggested decomposition in figure III-2.

Verification as a Sequence of Transformations

```
(NOT (OR (NOT (OR Input1 Input2))
         (NOT (OR Input3 Input4)) ))
              ↓  De-Morgan
(AND (NOT (NOT (OR Input1 Input2)))
     (NOT (NOT (OR Input3 Input4))) )
              ↓  Remove-Double-Negation
(AND (OR Input1 Input2)
     (NOT (NOT (OR Input3 Input4))) )
              ↓  Remove-Double-Negation
(AND (OR Input1 Input2)
     (OR Input3 Input4))
```

In summary, we have now cast the problem of verification as one of a *search* for a sequence of transformations, which will produce the original specification when applied to the composed specification.****

***Truth tables is a simple example of a scheme that could be used to verify the above assertion, but which does not meet the above requirement.

****Although no mention of it will be made in this paper, we are assuming that this search can be controlled by some appropriate search strategy, such as means-ends analysis.

E. Step 3: Determining the Generalized Composed Specification

In order to infer a new problem-redurt ion method from the given training example, we need to compute both its precondition (the IF part), and its postcondition (the THEN part). If we regard the original and composed specifications as instances of the precondition and postcondition of the new method, then clearly, the next step consists in generalising these specifications; furthermore, as we mentioned earlier, we would like to constrain the generalisation using the proof of the correctness of the decomposition.

In this section we describe how, given the verification proof as a sequence of transformations, the composed specification may be generalised using a restricted version of a well-known technique, *constraint backpropagatton(CBP).* CBP is a technique for determining the domain of a sequence of operators that produces some constrained range of states |6|. In our case, transformations can be viewed as operators, but we have no constraint on the range except that it match any arbitrary functional specification, which we denote by **<any-func-spec>.** (since we are learning arbitrary problem-reduction methods, and not sequences that lead to "solved" states, as in (4j.) The domain of the sequence, in our case, is a generalised composed specification that will produce, upon application of the sequence, a corresponding generalised original specification.

The problem with using weakest precondition techniques to compute the domain of a sequence is that, in many cases, disjunctive expressions are produced. In particular, disjuncts arise whenever sequences contain operators that were applied to only a part of the expression representing the problem state. For example, in the verification sequence above, the first application of the Remove-Double-Negation transformation was to a subexpression matching its precondition. Disjunctive expressions cause two kinds of problems. First, since the number of disjuncts can grow exponentially in the length of the sequence, storing all of them is a non-trivial issue. Second, even if all the disjuncts could be stored, it is quite likely that some of the disjuncts represent initial situations in which the sequence is an inefficient one to use. (For example, we have observed this problem crop up when using goal regression to compute the weakest precondition of a plan; some of the disjuncts represent initial states in which the plan is a very inefficient one to apply.)

We illustrate below a simple solution to this problem that uses information regarding how transformations were applied (in particular, their *bindings),* from the verification of the training instance, to prune out some disjuncts during the CBP process. (For example, the expression (NOT (NOT (NOT (NOT <bool-fn>)))) is a valid domain description of a sequence of two Remove-Double-Negations, which is not generated in the CBP computation below.)

Computing the generalized Composed Specification

```
(AND (OR Input_1 Input_2)
     <bool-fn1> )

              ↓        Remove-Double-Neg^-1

(AND <bool-fn2>
     (NOT (NOT <bool-fn1>)) )

              ↓        Remove-Double-Neg^-1

(AND (NOT (NOT <bool-fn2>))
     (NOT (NOT <bool-fn1>)) )

              ↓  De-Morgan^-1

(NOT (OR (NOT <bool-fn2>)
         (NOT <bool-fn1>) ))
```

To begin with, the expression <bool-fnl> above is produced by intersecting the range of the first Remove-Double-Negation transformation (any boolean function) with the range of the entire sequence (<any-func-spec>). Installing this expression as the second argument to the AND expression (the original specification), as we have done above, reflects the *context* in which this transformation was used in the forward direction (during verification). In the first step above, we backpropagate <bool-fnl> over this transformation, keeping the expression surrounding it unchanged. Similar remarks hold for the occurrence of <bool-fn2> above, and for the backpropagation over the second transformation. Finally, backpropagating over De-Morgan's does not present a similar problem, since its range intersects *onto* the expression back propagated over the first two transformations.***** The generalised composed specification itself is of little use to us; what we really need are generalisations of the specifications of the submodules (in general, subproblems) that constitute the product-of-sums circuit in figure III-2. These generalised specifications are shown below.**

Generalised specifications of the submodules in figure III-2

```
P_1' : (EQUALS X (NOT <bool-fn2>) )
P_2' : (EQUALS Y (NOT <bool-fn1>) )
P_3' : (EQUALS <out> (NOT (OR X Y)) )
```

Comparing the generalised submodule specifications in figure III-1 with the submodule specifications given in figure III-2, we see that the important feature of the two submodule specifications P_1 and P_2 (the two input NOR gates in figure II1-2), which enabled the verification to carry through, is that they both be the *negation of some boolean function.*

F. Step 4: Determining the (Generalized Original Specification

In this section, we describe methods for generalizing the original specification, given that the generalized composed specification has already been computed. One simple method involves storing the variable bindings generated while computing the generalised composed specification (for example, the subexpression (OR $Input_3$ $Input_4$) was replaced by <bool-fnl>), and applying these substitutions to the original specification. Another method, which illustrates better that the composed and original specifications form the domain and range of a sequence of transformations, involves, as we show below, reapplying the transformation sequence to the generalised composed specification.

***** Although Implementing this procedure has been an easy task, attempts at formalising it have not yet been successful.

**Figuring out the generalisation of each submodule specification from the generalised composed specification is straightforward, provided some book-keeping was done while forming the composed specification in the first place.

Computing the Generalised Original Specification

```
(NOT (OR (NOT <bool-fn2>)
         (NOT <bool-fn1>) ))
                    ↓ De-Morgan
(AND (NOT (NOT <bool-fn2>))
     (NOT (NOT <bool-fn1>)) )
                    ↓ Remove-Double-Negation
(AND <bool-fn2>
     (NOT (NOT <bool-fn1>)) )
                    ↓ Remove-Double-Negation
(AND <bool-fn2> <bool-fn1>)
```

Comparing the generalised original specification - the last expression in the above sequence - with the original specification P in figure II1-2, we see that a generalization of the original specification has been achieved from a conjunction of disjunctions to a *conjunction of any boolean functions.*

G. Step 5: Forming the New Problem-Reduction Method

We showed above how the original specification P and the submodule specification P. could be generalized The final step is to form a new problem-reduction method that is based on these generalised specifications. It is clear that the generalized original specification will form the precondition, or LHS, of the new problem-reduction method. Also, the postcondition, or RHS, of the new method can be formed from the generalized submodule specifications and the combinator C. For our present example, the new problem-reduction method that is inferred (from the training instance in figure III-2) is the one given in figure III-1.

IV Learning Problem-Reduction Methods for Symbolic Integration

In this section we will briefly illustrate how the same VBL technique can be used to learn problem-reduction methods in symbolic integration. This will provide some justification for our claim that VBL is general technique, and can be applied to more than one domain. Further details of the steps sketched below are described in [1S|.

A. Symbolic Integration as Problem-Reduction

We begin by viewing the process of integration in terms of problem-reduction. To this end, note that the language of instances of problems in symbolic integration is the language of integrals of mathematical functions. The language of generalizations of problems includes the ability to specify *arbitrary* functions as part of integrals. Problem-reduction methods here are the standard rules of integration *** We may thus view the problem of learning integration methods from examples, as an instance of the more general problem of inferring problem-reduction methods from examples.

B. An Illustration of Problem-Reduction in Symbolic Integration

Consider the following example of a problem in Symbolic Integration, and a way of decomposing the problem.

Original Problem P: $\int\{3x^2 + sin(x)\}dx$

Decomposition: $\int 3x^2dx + \int sin(x)dx$

***Such as the one we described in section I.B.

The original problem P has been decomposed above into two sub-problems, P_1, which is $\int 3x^2dx$, and p_2 , which is $\int sin(x)dx$. The combinator C above is addition The generalisation problem here lies in determining a general method, by generalising the specifications P and P., which could produce the above decomposition. In the next few paragraphs, we summarise the main steps involved in inferring the general method, $\int\{f_1(x) + f_2(x)\}dx \rightarrow \int f_1(x)dx + \int f_2(x)dx$, from the above training instance.

C. Steps1&2: Forming the Composed Specification and Verification

Here, unlike the circuits domain, we do not need to construct the composed specification, since the combinator (addition) specifies that explicitly. We can proceed directly to the verification step.

We now describe what verification means in symbolic integration. We can make use of the following result, which asserts that if the derivatives of two integration problems are shown to be equal, then the solutions to the two problems are identical. (The reasoning is that if the derivatives of two functions are equal, then they differ at most by a constant. Two solutions to an indefinite integration problem which differ by a constant are both instances of a more general family of solutions, F(x) + a constant.) We may thus take as the problem of verification in symbolic integration, the task of showing that the *derivative* of the composed specification is equal to the *derivative* of the original specification. Given the above result, this will imply the equivalence of the original specification and the composed specification. As earlier, we can proceed to verify instances of problem-reductions using a sequence of transformations, except in this case transformations correspond to *rules of differentiation.* An example of such a transformation, which would be useful in verifying the above problem-reduction instance, is given below.

Derivative-of-a-Sum

Precondition:	Postcondition:
$d/dx\{f1(x)+f2(x)\}$	$d/dx\{f1(x)\} + d/dx\{f2(x)\}$

D. Step S: Determining the General Composed Specification

Given a verification of a problem-reduction in symbolic integration as a sequence of transformations of the kind shown above, we can use a procedure completely analogous to the one described earlier in section 3 5 to determine the generalized composed specification As earlier, we have to restrict the CBP procedure in order to avoid generating disjuncis sing this procedure, we will obtain the following generalized composed specification, $\int f_2(x)dx + \int f_1(x)dx$ Comparing this with the composed specification $\int 3x^2dx + \int sin(x)dx$, we see the generalization that has been achieved.

£. Step 4: Determining the Generalised Original Specification

Having obtained the generalized composed specification, the generalised original specification can be determined, once again, in either of the two ways described in section 3.6. That is, we may use the bindings obtained during the CBP computation of the generalised composed specification, or reapply the sequence of transformations used in the verification to the generalised composed specification. Using either of these two ways, we will obtain the following generalised original specification, $\int\{f_1(x) + f_2(x)\}dx$.

F. Step 5: Forming the new Integration method

Finally, given the generalised composed and generalised original specification as above, we can form the new integration method by defining the latter to be its precondition and the former to be its postcondition. This is shown below.****

New Method:

Precondition: $\int \{f_2(x) + f_1(x)\}dx$

Postcondition: $\int f_2(x)dx + \int f_1(x)dx$

V Conclusions

A. Different Perspectives for viewing VBL

We begin summarizing the VBL technique by viewing it from a number of different perspectives.

VBL as learning problem-solving methods: VBL can be described as a general technique for learning problem-solving methods. As it is an analytical generalization technique, one of its nice features is that it produces *justifiable* generalizations |4|, as opposed to *empirical* generalization techniques, such as described in [14], which rely primarily on detecting *syntactic* similarities among training instances.

VBL at forming macros: Another way of thinking about what VBL does is by viewing each problem-reduction method as being constructed as a *macro* of the sequence of transformations used in the verification. In this sense VBL seems similar to earlier work on plan generalization systems like STRIPS [3], but operating in the problem-reduction space, as opposed to the state-space. On the other hand, it is important to note that any proof technique for generating verifications can be used (provided, of course, that it meets the requirement that we imposed in section 3.4), and using a sequence of transformations is just one such scheme.

VBL at learning plant Circuit design may be viewed as a planning problem. From this perspective each submodule becomes a planning method, and a circuit becomes a network of such methods. Given a particular plan used in a specific situation, one can generalize it by generalizing the class of situations in which exactly the same plan could be applied, which is what MACROPS did |3|. |7] shows how this may be done for circuit design. This highlights an interesting new feature of VBL, which is the ability to *generalize plans by generalizing the individual methods in the plan.* For example, in section 3, the specification of the NOR gate was generalized to a *negation of any boolean function,* which really represents a class of possible submodules (for eg., a NAND gate). Thus, we can describe VBL as a technique for generalizing plans by generalizing the subgoals achieved by the individual methods constituting a given plan.

B. Limitations of the technique

We must now make clear some important requirements that need to be fulfilled in order to apply VBL to some given domain. We summarize below some of these requirements. [7] describes, in more detail, problems that are anticipated in using this and other related techniques in real-world situations.

Underlying Domain Theory : The success of the VBL technique rests on its capability to construct a proof of the correctness of a decomposition, which implies the existence of a *strong* underlying theory of the domain (such as *Circuit Analysis).* Most of the theory that is needed, for example, the transformations and the ability to propagate constraints, we assume are part of a know ledge-based system *on top* of which the Learning Apprentice will be designed. While this is true for the specific Learning Apprentice system LEAP, there are domains, such as *Well-Log Interpretation,* where the lack of a *strong* underlying domain model prevents the successful application of pure analytical generalisation methods [15]. This in turn motivates the need for an *empirical* component in the overall learning system.

Adequacy of the Verification Technique: One important question, which directly affects the usefulness of VBL, is - how hard is it to verify arbitrary problem decompositions? It might be that a very large number of transformations are needed to cover a wide range of problems. One related issue, which we have not discussed in this paper, is the existence of *multiple ways of generating verification proofs* For example, an alternative scheme for verifying the product-of-sum circuit example is that of *truth trees [*16], which has certain desirable properties (such as guaranteed termination), making the issue of control an easy one; however, these properties hold only for the restricted class of *combinational* circuits. More generally, we believe that constructing verification proofs as a sequence of transformations is a weak but general way of approaching the verification problem. By exploiting the properties of the particular domain (such as the functional specifications of the product-of-sums circuit being *boolean* expressions), we may be able to come up with more powerful (and more restricted) schemes for verification.

C. Fntnre Research Topics

We now describe some avenues for further research that we are currently exploring.

Extending Weakest Precondition Techniques: One of the problems that we described earlier had to do with the appearance of disjunctive expressions during the CBP computation Some of the disjuncts, we said, correspond to initial situations in which it is *correct,* but not *efficient,* to use the sequence. It seems that we need more powerful analytical tools in order to learn the class of situations in which it is efficient to use a macro (as opposed to when it *can* be used). Work is in progress on formulating this problem within the framework of *Goal-Directed Learning* |4), by providing the system with an explicit definition of the "class of states in which it is efficient to apply a macro", and having this aid the generalization process.

Extending the notion of Verification: Although we have formulated the problem of verification as one of finding a sequence of transformations, it is important to note that the concept of verification is more general, and, in the extreme, it could be viewed as a restricted form of *theorem-proving,* (the assertion being proved is the invariance property of problem-reduction methods.) Also, the connection between work on proving circuits correct, and that of generalizing problem-reduction methods seems an interesting one to pursue. In particular, we need to state more precisely the additional requirement that we imposed on proof techniques for verification - that they focus on those features that were important in order for the proof to work.

****Note that this is exactly the Integral-of-a-sum method that we Illustrated in section I.B.

VI Acknowledgments

1 thank the following people for their contributions to the form and content of this paper Tom Mitchell, for many ideas and for greatly improving the presentation of the paper. Saul Amarel and Natesa Sridharan, for their careful reading of a draft. Lou Steinberg and other members of the Rutgers AI/VLSI project, for many useful discussions. Smadar Kedar-Cabelli, Prasad Tadepalli and Rich Keller, for many interesting discussions on analytical learning techniques.

References

1 Mitchell, T, Steinberg, L and Shulman, J "A Knowledge-Based Approach to Design" In *IEEE Principles of Knowledge-Based Systems* Denver. December, 1984.

[2] Barstow, D "An Experiment in Knowledge-Based Automatic Programmin g." *Artificial Intelligence* 12.2 (1979) 73-119

[3) Fikes, R., Hart, P and Nilsson, N "Learning and Executing Generalized Robot Plans." *Artifical Intelligence.* 3 (1972) 251-288.

|4| Mitchell, T. "Learning and Problem-Solving " In IJ*CAI-83*. August, 1983, 1139-1151

|5| Minton, S "Constraint-Based Generalization " In *AAAI-84.* Austin, Texas, August, 1984, 251-254

|6] Utgoff, P. and Mitchell, T. "Acquisition of Appropriate Bias for Inductive Concept Learning." In *AAA1-82.* August, 1982, 414-417.

|7] Mitchell, T., Mahadevan, S. and Steinberg, L., "LEAP - A Learning Apprentice for VLSI Design", to appear in *IJCA1-85*

8] Amarel, S., "An Approach to Heuristic Problem-Solving and Theorem-Proving in the Propositional Calculus." In *Systems and Computer Science.* Univ. of Toronto Press, 1967, .

9] Nilsson, N. *Problem-Solving Methods in Artificial Intelligence.* McGraw-Hill, 1971.

10] deJong, G. "Automatic Schema Acquisition in a Natural Language Environment." In *AAAI-82.* Pittsburgh, 1982, 410-413.

11] Silver, B. "Learning Equation Solving Methods from Examples" In *IJCA1-88* August, 1983, 429-431.

12] Barrow, H. "Proving the Correctness of Digital Hardware Designs." In *AAAI-88.* Washington D C , 1983, 17-21.

13] Mahadevan, S. "Verification-Based Learning: A Generalization Strategy for Inferring Problem-Reduction Methods.", Technical report LCSR-TR-66, Rutgers Univ, 1985.

14] Porter, B. and Kibler, D. "Learning Operator Transformations." In *AAAI-84.* Austin, Texas, August, 1984, 278-282.

15] Smith, R G , Winston, H.A., Mitchell, T M., and Buchanan, B.C., "Representation, Use and Generation of Explicit Justifications for Knowledge Base Refinement", submitted to *IJCA185*

16] Jefferey, R. . *Formal Logic: Its Scope and Limits.* McGraw-Hill, 1967.