MCC Technical Report No. ACA/AI-CAD-079-88

Alternative Explanation Structures for Explanation-Based Learning

Ramon D. Acosta and Michael N. Huhns

March 1988

Abstract

Explanation-based learning (EBL) is a knowledge-intensive analytic technique by which learning systems can capture and generalize problem-solving experience from single training examples. Essential to the EBL technique is the explanation structure used for describing how the training example is solved. The explanation structure used in most implementations of EBL is a proof tree that captures dependencies among the literals of the rules involved in finding the solution. Unfortunately, use of proof trees as explanations restricts generalizations to being structurally equivalent to their respective training examples. This paper presents alternative explanation structures that allow EBL systems to learn more interesting problem-solving features. These explanation structures include chronologically ordered rules, partial chronologically ordered rules, and rule-dependency graphs. We conclude by showing how viewing explanations as rule-dependency graphs can be of benefit to systems that learn abstractions of training examples suitable for analogical reasoning.

Microelectronics and Computer Technology Corporation
Advanced Computer Architecture
Artificial Intelligence Laboratory
3500 West Balcones Center Drive
Austin, TX 78759
(512) 343-0978

1 Introduction

Explanation-based learning (EBL) is an analytic knowledge-intensive technique by which a machine learning system can capture and generalize problem-solving experience from a single training example. The applicability of EBL seems very general; successful implementations range from theorem proving systems to general-purpose planners. Because of this initial success, variations and extensions of EBL are currently being investigated by a number of machine learning researchers [4, 7, 12, 15, 17, 18, 20, 21].

Most implementations of EBL conform to the following methodology: given a problem (*training example*) to be solved and *domain knowledge*, a performance system is used to find a solution (*goal*). In the process, an *explanation* describing why the solution solves the problem is recorded in terms of the rules applied by the performance system. Subsequently, an *explanation structure*, corresponding to the uninstantiated form of the domain rules in the explanation, is generalized in such a way that sufficient constraints on the domain of training examples for which the explanation holds are computed. Finally, a generalization of the training example and its solution is extracted from the generalized explanation.

In its typical implementation, EBL can be viewed as simply compiling an explanation for a training example, based on rule instance invocation, into a generalized rule, known as a *macrorule*. The expectation is that subsequent application of this macrorule will be more efficient than applying the original form of the knowledge, thus reducing the search space for finding a solution. Interestingly, what is actually compiled is the control knowledge related to searching and applying rule instances in the training example. An additional EBL requirement is that the antecedents of a learned macrorule must be *operational*, *i.e.*, in terms of attributes that are readily observable in problem descriptions [13, 14, 20].

Of obvious relevance to how macrorules are computed is the type of structure used to capture the explanation. EBL systems described in the literature have typically employed a causally connected proof tree having edges between individual antecedent and consequent instances of dependent rules. Although techniques for manipulating the explanation before computing a macrorule have been proposed, such as pruning hierarchical inheritance (ISA) rules [4, 21], the structural constraints imposed by using a proof tree ultimately result in only one macrorule being computed per explanation.

This paper describes alternative explanation structures that allow several useful macrorules to be computed from a single training example. These structures use rule instances, instead of antecedent and consequent instances, to capture explanations. Three alternatives are presented: chronological-precedence graphs, partial chronological-precedence graphs, and rule-dependency graphs. In particular, we have experimented with recording explanations as rule-dependency graphs in the Argo system [1, 9, 10, 11], and using abstractions of these graphs to compute abstractions of macrorules. The macrorule abstractions are suitable for use in a type of derivational analogy [2, 3].

2 Inferencing Model

Although not essential to most of this presentation, it is assumed that rule-based forward chaining is employed in solving problems and, thus, in formulating explanations for training examples. The view adopted is that of a production system, characteristic of many so-called expert systems, in which rules are successively applied to effect changes to a working memory of assertions [8].

Each rule, alternatively referred to as an operator or production, consists of a *precondition* (*antecedents*) and a *postcondition* (*consequents*). Both the antecedents and consequents are comprised of positive conjunctive *literals* (*atomic formulas*) in first-order logic. The following notation is used:

$$R_i = (A_i,\ C_i)$$

$$A_i = \{a_{i1}, a_{i2}, \ldots, a_{iL_i}\}$$

$$C_i = \{c_{i1}, c_{i2}, \ldots, c_{iK_i}\}$$

where R_i is a rule whose antecedents and consequents are the sets A_i and C_i, respectively. A literal consists of a predicate and a list of one or more *terms*, also called *arguments*. Terms can be constants, functions, or universally quantified variables.

A rule is a problem-reduction operator whose effect, when applied to a working memory of problem assertions, is that assertions unifying with the rule's antecedents are deleted from the working memory, while those generated by the rule's consequents are added as new assertions to the working respectively, of the proof.

The definitions for explanation, explanation structure, and generalized explanation structure used in Figures 2–4 are essentially the same as the terms suggested in [21]. In particular, the explanation structure is a non-generalized representation of the solution in terms of domain knowledge that is standardized apart. Thus, the explanation structure, while arising as a result of solving the training example, does not refer to training example knowledge.

As mentioned, an important requirement for EBL systems is that a learned macrorule must have operational antecedents [20]. Most systems take a simplified view towards this requirement: since the proof succeeds for the training example, computing a macrorule by generalizing the proof preserves operationality. Work has been carried out in formalizing and applying the notion of operationality to improve learning capabilities [13, 14]. Although operationality in EBL formulations is considered important, for the most part it is not examined closely in this paper in order to limit the scope of the discussion.

The generalization technique proposed in [4], called Explanation Generalization using Goal Substitution (EGGS) in [21], suggests that simply having a literal-dependency graph (LDG) is sufficient for computing a generalization. The LDG is a directed-acyclic graph having as nodes the antecedent and consequent instances of the rules applied in solving the training example. A directed edge from a consequent node to an antecedent node in the LDG indicates causality. That is, in solving the training example, an assertion added to working memory as a result of executing the consequent node's rule is subsequently used to satisfy part of the precondition of the antecedent node's rule. Thus, the LDG maintains logical dependencies between consequent and antecedent instances unified in solving the training example. The literal-dependency graph for the `SAFE-TO-STACK` example is illustrated in Figure 6.

The EGGS algorithm, outlined in Figure 7, can be applied to the LDG to obtain the macrorule in Figure 5. A general substitution, σ, is obtained by accumulating the substitutions for unifications of consequent and antecedent nodes for all edges in the LDG. The macrorule antecedents are obtained by applying σ to antecedent instances having no incoming edges in the LDG. Similarly, consequents of the macrorule consist of consequent instances having no outgoing edges in the LDG to which σ has been applied.

Domain Knowledge

```
R1 = ({VOLUME(?p1, ?v1), DENSITY(?p1, ?d1)},
      {WEIGHT(?p1, ?v1*?d1)})

R2 = ({ISA(?p1, ENDTABLE)},
      {WEIGHT(?p1, 5)})

R3 = ({WEIGHT(?p1, ?w1), WEIGHT(?p2, ?w2), LESS(?w1, ?w2)},
      {LIGHTER(?p1, ?p2)})

R4 = ({LIGHTER(?x, ?y)},
      {SAFE-TO-STACK(?x, ?y)})   (goal)

LESS(0.1, 5)

⋮
```

Training Example

```
ON(OBJ1, OBJ2)
ISA(OBJ1, BOX)
ISA(OBJ2, ENDTABLE)
COLOR(OBJ1, RED)
COLOR(OBJ2, BLUE)
VOLUME(OBJ1, 1)
DENSITY(OBJ1, 0.1)

⋮
```

Figure 1: Knowledge for the SAFE-TO-STACK Example

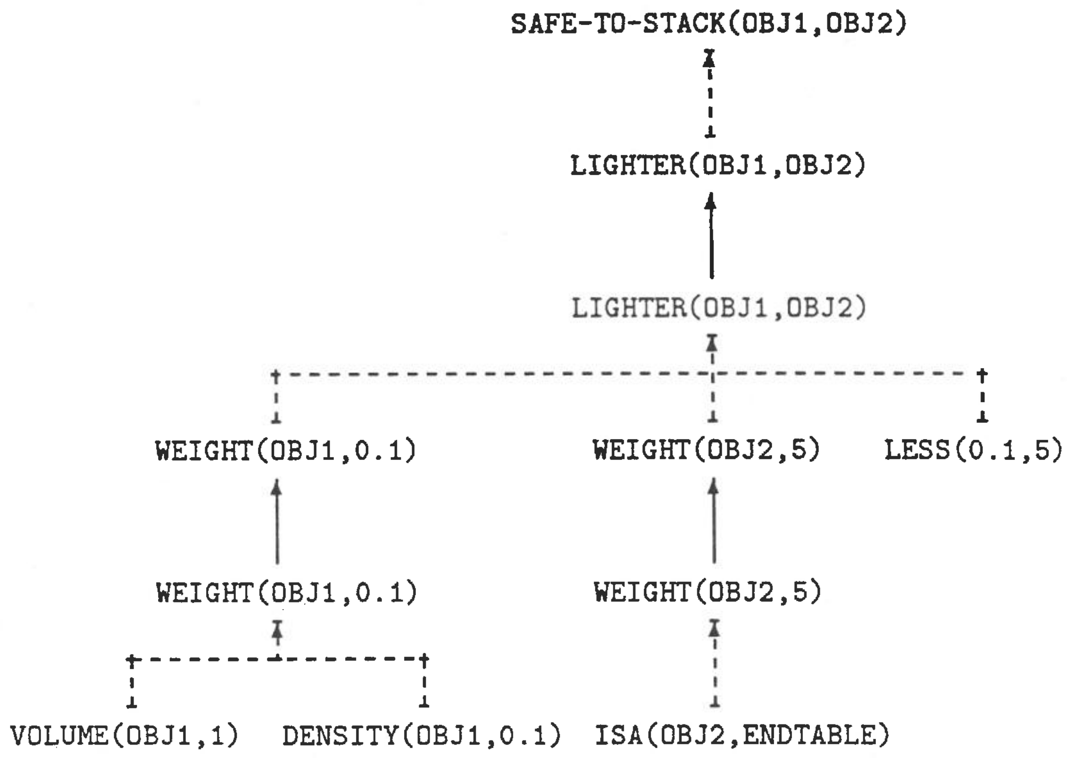


Figure 2: Explanation (Instantiated Proof Tree) for the `SAFE-TO-STACK` Example

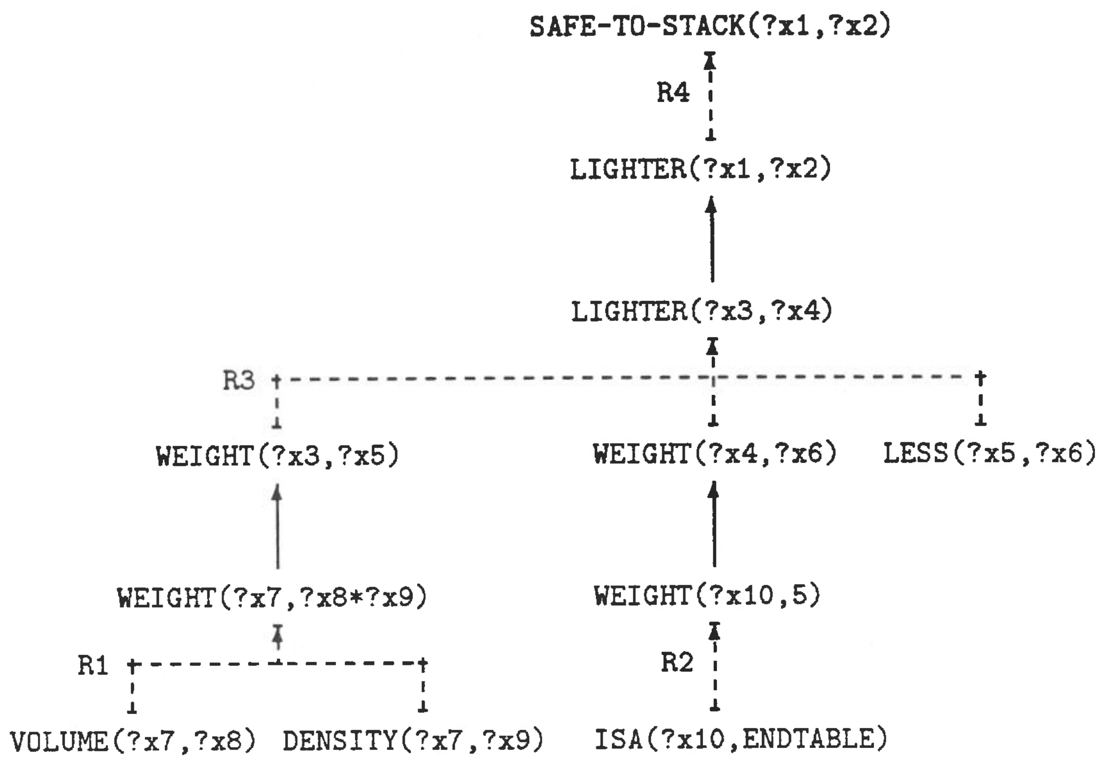


Figure 3: Explanation Structure for the SAFE-TO-STACK Example

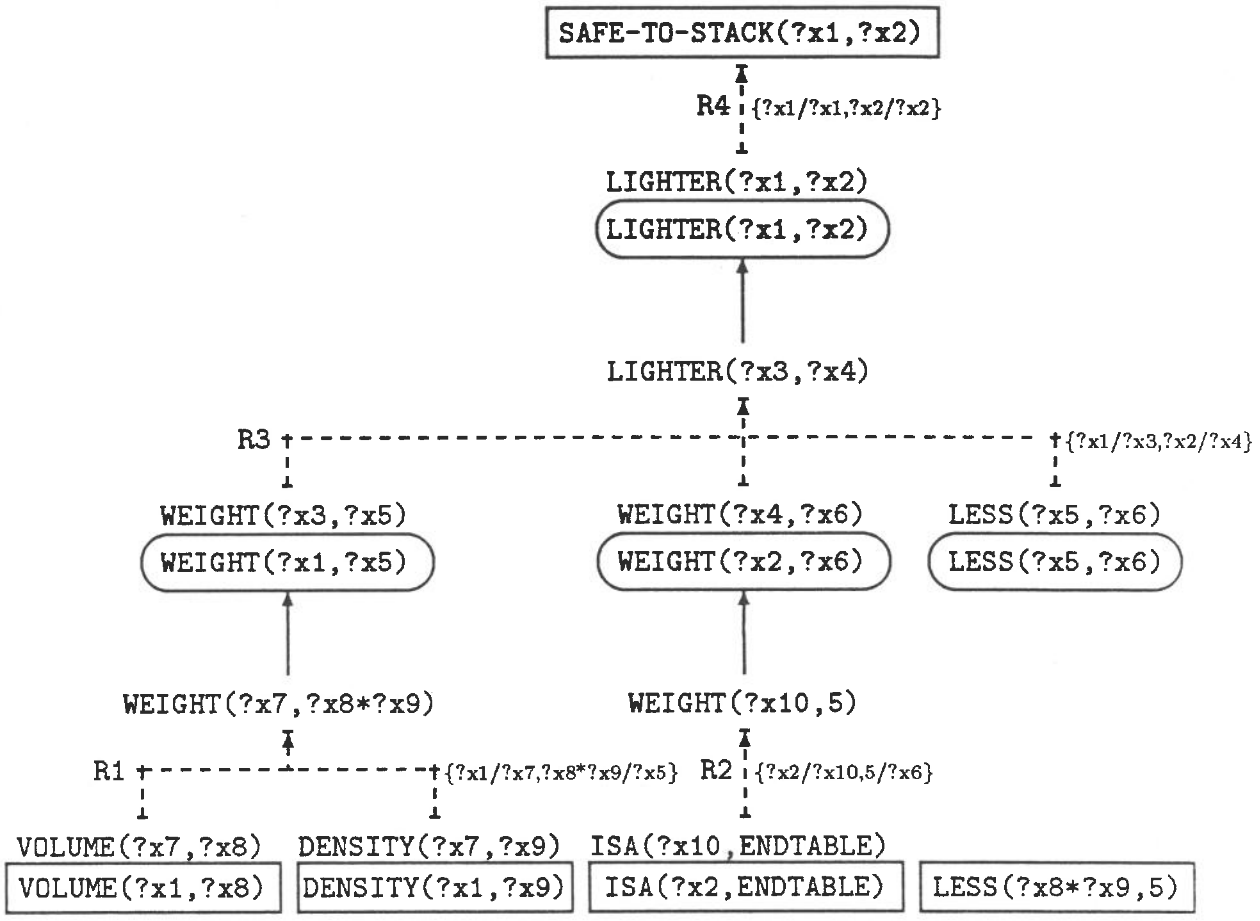


Figure 4: Generalized Explanation Structure for the SAFE-TO-STACK Example

```
Rmacro = ({VOLUME(?x1, ?x8),
           DENSITY(?x1, ?x9),
           ISA(?x2, ENDTABLE),
           LESS(?x8*?x9, 5)},
          {SAFE-TO-STACK(?x1, ?x2)})
```

Figure 5: Macrorule for the SAFE-TO-STACK Example

$$\begin{tikzcd}
c_{41} = \texttt{SAFE-TO-STACK(?x1,?x2)} & \\
a_{41} = \texttt{LIGHTER(?x1,?x2)} & \\
c_{31} = \texttt{LIGHTER(?x3,?x4)} \arrow[u] & \\
a_{33} = \texttt{LESS(?x5,?x6)} & \\
a_{31} = \texttt{WEIGHT(?x3,?x5)} & a_{32} = \texttt{WEIGHT(?x4,?x6)} \\
c_{11} = \texttt{WEIGHT(?x7,?x8*?x9)} \arrow[u] & c_{21} = \texttt{WEIGHT(?x10,5)} \arrow[u] \\
a_{12} = \texttt{DENSITY(?x7,?x9)} & a_{21} = \texttt{ISA(?x10,ENDTABLE)} \\
a_{11} = \texttt{VOLUME(?x7,?x8)} &
\end{tikzcd}$$

Figure 6: Literal-Dependency Graph for the SAFE-TO-STACK Example

```
(1) For each $edge(c_{ik}, a_{jl})$ in the LDG
    (1a) $\sigma$ := $\sigma \cup$ $most\text{-}general\text{-}unifier(c_{ik}\sigma, a_{jl}\sigma)$;
(2) Rmacro := ($\{a_{jl}\sigma \mid a_{jl}$ has no incoming edges in the LDG$\}$,
             $\{c_{ik}\sigma \mid c_{ik}$ has no outgoing edges in the LDG$\}$).
```

Figure 7: Macrorule Computation Using the LDG

Although [4] suggests computing the general substitution as the performance system solves a training example, this is by no means necessary as long as an LDG is recorded as part of the problem-solving process. Moreover, performance considerations might make it undesirable to have the extra unifications required for building σ be an inherent part of the problem solver. In any case, the issue of when to compute macrorules is an orthogonal consideration if the procedure in Figure 7 for using LDGs as explanation structures is employed.

In STRIPS [6, 7] a procedure similar to EGGS is employed to generalize problem-solving plans. The main difference between the STRIPS procedure and that in Figure 7 is that instead of cumulatively building up a σ for all LDG edges, *most-general-unifiers* for all edges in the LDG are progressively applied to all literal instances in the proof. Although intuitively less efficient than EGGS, generalization in STRIPS is guaranteed to produce the same macrorules obtained with either EBG or EGGS.

Two explanation structures have been described: the (uninstantiated) proof tree and the literal-dependency graph. Note that the proof tree contains the same information as the LDG, plus additional information linking antecedent and consequent instances to the rule instances associated with them. The rule instance information is necessary for computing macrorules using goal regression, as described in [20]. The LDG has enough information for computing macrorules using the general substitution technique described in [4]. Generalizing either of these explanation structures results in computing the same macrorule. Again, only one macrorule is computed, and it is applicable only to problems that are structurally equivalent to the training example.

The following argument illustrates this last point. Consider the macrorule in Figure 5. This compiled knowledge to determine when it is `SAFE-TO-STACK` one object on another is applicable *only* to problems in which the weight of the object bound to `?x1` is calculated using its volume and density, the object bound to `?x2` is an `ENDTABLE`, and the weight of the `?x1` object is less than 5, the default weight of `ENDTABLE`s. There are similar problems that can be solved by *exactly the same* domain and control knowledge as that used to solve the training example, but that *cannot* be solved by the macrorule computed from the training example. Given the training example, it would clearly be useful to learn macrorules for solving more general problems about instances in which it is `SAFE-TO-STACK` objects, for example, when the `?x1` object is heavier than an ENDTABLE. Thus, a need arises to consider alternative explanation structures capable of leading to macrorules for problems differing structurally from the training example.

4 Chronological-Precedence Graphs

The main limitation of the schemes mentioned above in terms of number of macrorules per training example has to do with the fact that dependency information is recorded at the granularity of literals. This results in only one literal-dependency graph for each training example. We now present several explanation structures for overcoming this limitation that use rules, instead of literals, as the grain size for representing dependencies in domain knowledge applied to training examples. These alternatives yield more than one LDG and, thus, allow learning more than one macrorule for a given training example.

A simple approach to expressing an explanation is as a chronologically ordered set of rule instances—totally ordered according to when the rules are applied in solving the training example. This set can be represented using a directed-acyclic graph having rule instances as vertices and directed edges denoting the precedence relation among the instances. We call this a chronological-precedence graph (CPG). Figure 8 contains the CPG for the SAFE-TO-STACK example. The CPG information, not being based on fine-grained logical dependencies between rule literals, can lead to potentially many literal-dependency graphs and, consequently, many different macrorules.

Conceptually, macrorules can be computed by combining literal dependency graphs for chronologically dependent rule instances in the CPG, as in Figure 9. In this figure, a *consistent* LDG is characterized as follows: 1) consequent-instance nodes have at most one outgoing edge, 2) antecedent-instance nodes have at most one incoming edge, and 3) a general substitution σ exists. The latter is obtained by cumulatively unifying consequent/antecedent edges in a manner similar to step (1) of Figure 7. A simple optimization used in implementing this procedure is to maintain the σ substitution for each LDG in $\texttt{subLDG}_{ij}$. This optimization simplifies the computation of merged-LDGs in step (2) by replacing the need to carry out cumulative edge unifications with a relatively simple check to see if the LDG

$$\begin{tikzcd}
R1 \arrow[d] \\
R2 \arrow[d] \\
R3 \arrow[d] \\
R4
\end{tikzcd}$$

Figure 8: Chronological-Precedence Graph for the SAFE-TO-STACK Example

substitutions are compatible. Additionally, step (3) of Figure 8 is typically unnecessary since macrorules can be trivially computed as a side effect of building merged-LDGs.

```
(1) For each $path(R_i, R_j)$ in the CPG

    (1a) subLDG$_{ij}$ := set of all consistent LDGs having edges from
                     consequents in $R_i$ to antecedents in $R_j$;

(2) merged-LDGs := set of consistent LDGs for the entire CPG
                   obtained by merging all LDG combinations
                   from the subLDG$_{ij}$ combinations;

(3) For each LDG in merged-LDGs, compute a macrorule using
    the procedure in Figure 7.
```

Figure 9: Macrorule Computation Using the CPG

Use of the CPG as the explanation structure for the SAFE-TO-STACK example leads to the macrorules shown in Figure 10. In addition to Rmacro13, which is the same as Rmacro in Figure 5, 13 additional macrorules are computed. With the exception of Rmacro13 and Rmacro14, however, these macrorules are not operational with respect to the training example. For example, they require having assertions about WEIGHTs and LIGHTER, features that would typically not be directly observable. Thus, a single training example has generated 14 macrorules, of which twelve are unusable.

The problem with using the CPG is that not enough constraints are placed on dependencies between rule instances. The only causal information recorded is that if R_i chronologically precedes R_j, then logical dependencies involving consequents of R_i and antecedents of R_j are possible, but not vice versa. Consequently, LDGs for $O(n^2)$ rule dependencies, the number of edges in a CPG's transitive closure, must be considered in step (1) of Figure 9. Coupled with the multiple LDGs that are possible for a given pair of dependent rules in step (1a), a combinatorial explosion of the number of computed macrorules, most of which are unusable, can result.

Again, the aim is to generate macrorules that differ structurally from the training example, yet result in compiled knowledge that will be useful for solving future problems. An obvious way to reduce the number of unusable rules generated from the CPG is by constraining rule dependencies in such a way as to reflect more closely the control knowledge used for the training example.

5 Partial Chronological-Precedence Graphs

Using the fact that some rules decompose problems into independent subproblems, one can envision an explanation structure consisting of a partial-chronological-precedence graph (PCPG) of rules. For example, if R_1 decomposes a problem into independent subproblems A and B, then R_1 chronologically precedes all of the rules used in solving A and B. There is, however, no chronological relation between the rules for solving A and those for solving B; they do not need to be considered for step (1) of Figure 9.

Consequently, the PCPG is more constrained than the CPG and fewer $\texttt{subLDG}_{ij}$ sets are possible. Unfortunately, within a subproblem, rule dependencies in the PCPG continue to be based upon chronology. As with the CPG, this chronological basis allows the possibility for generating many useless macrorules.

```
Rmacro1 =  ({VOLUME(?x7, ?x8), DENSITY(?x7, ?x9), ISA(?x10, ENDTABLE),
             WEIGHT(?x3, ?x5), WEIGHT(?x4, ?x6), LESS(?x5, ?x6),
             LIGHTER(?x1, ?x2)},
            {WEIGHT(?x7, ?x8*?x9), WEIGHT(?x10, 5),
             LIGHTER(?x3, ?x4), SAFE-TO-STACK(?x1, ?x2)})
Rmacro2 =  ({VOLUME(?x7, ?x8), DENSITY(?x7, ?x9), ISA(?x10, ENDTABLE),
             WEIGHT(?x1, ?x5), WEIGHT(?x2, ?x6), LESS(?x5, ?x6)},
            {WEIGHT(?x7, ?x8*?x9), WEIGHT(?x10, 5), SAFE-TO-STACK(?x1, ?x2)})
Rmacro3 =  ({VOLUME(?x3, ?x8), DENSITY(?x3, ?x9), ISA(?x10, ENDTABLE),
             WEIGHT(?x4, ?x6), LESS(?x8*?x9, ?x6), LIGHTER(?x1, ?x2)},
            {WEIGHT(?x10, 5), LIGHTER(?x3, ?x4), SAFE-TO-STACK(?x1, ?x2)})
Rmacro4 =  ({VOLUME(?x4, ?x8), DENSITY(?x4, ?x9), ISA(?x10, ENDTABLE),
             WEIGHT(?x3, ?x5), LESS(?x5, ?x8*?x9), LIGHTER(?x1, ?x2)},
            {WEIGHT(?x10, 5), LIGHTER(?x3, ?x4), SAFE-TO-STACK(?x1, ?x2)})
Rmacro5 =  ({VOLUME(?x7, ?x8), DENSITY(?x7, ?x9), ISA(?x3, ENDTABLE),
             WEIGHT(?x4, ?x6), LESS(5, ?x6), LIGHTER(?x1, ?x2)},
            {WEIGHT(?x7, ?x8*?x9), LIGHTER(?x3, ?x4), SAFE-TO-STACK(?x1, ?x2)})
Rmacro6 =  ({VOLUME(?x7, ?x8), DENSITY(?x7, ?x9), ISA(?x4, ENDTABLE),
             WEIGHT(?x3, ?x5), LESS(?x5, 5), LIGHTER(?x1, ?x2)},
            {WEIGHT(?x7, ?x8*?x9), LIGHTER(?x3, ?x4), SAFE-TO-STACK(?x1, ?x2)})
Rmacro7 =  ({VOLUME(?x1, ?x8), DENSITY(?x1, ?x9), ISA(?x10, ENDTABLE),
             WEIGHT(?x2, ?x6), LESS(?x8*?x9, ?x6)},
            {WEIGHT(?x10, 5), SAFE-TO-STACK(?x1, ?x2)})
Rmacro8 =  ({VOLUME(?x2, ?x8), DENSITY(?x2, ?x9), ISA(?x10, ENDTABLE),
             WEIGHT(?x1, ?x5), LESS(?x5, ?x8*?x9)},
            {WEIGHT(?x10, 5), SAFE-TO-STACK(?x1, ?x2)})
Rmacro9 =  ({VOLUME(?x7, ?x8), DENSITY(?x7, ?x9), ISA(?x1, ENDTABLE),
             WEIGHT(?x2, ?x6), LESS(5, ?x6)},
            {WEIGHT(?x7, ?x8*?x9), SAFE-TO-STACK(?x1, ?x2)})
Rmacro10 = ({VOLUME(?x7, ?x8), DENSITY(?x7, ?x9), ISA(?x2, ENDTABLE),
             WEIGHT(?x1, ?x5), LESS(?x5, 5)},
            {WEIGHT(?x7, ?x8*?x9), SAFE-TO-STACK(?x1, ?x2)})
Rmacro11 = ({VOLUME(?x3, ?x8), DENSITY(?x3, ?x9), ISA(?x4, ENDTABLE),
             LESS(?x8*?x9, 5), LIGHTER(?x1, ?x2)},
            {LIGHTER(?x3, ?x4), SAFE-TO-STACK(?x1, ?x2)})
Rmacro12 = ({VOLUME(?x4, ?x8), DENSITY(?x4, ?x9), ISA(?x3, ENDTABLE),
             LESS(5, ?x8*?x9), LIGHTER(?x1, ?x2)},
            {LIGHTER(?x3, ?x4), SAFE-TO-STACK(?x1, ?x2)})
Rmacro13 = ({VOLUME(?x1, ?x8), DENSITY(?x1, ?x9),
             ISA(?x2, ENDTABLE), LESS(?x8*?x9, 5)},
            {SAFE-TO-STACK(?x1, ?x2)})
Rmacro14 = ({VOLUME(?x2, ?x8), DENSITY(?x2, ?x9),
             ISA(?x1, ENDTABLE), LESS(5, ?x8*?x9)},
            {SAFE-TO-STACK(?x1, ?x2)})
```

Figure 10: Macrorules for the SAFE-TO-STACK Example Based on the CPG

6 Rule-Dependency Graphs

Additional constraints can be placed on the explanation structure by considering logical rule dependencies. Thus, in the development of Argo [1, 9, 10, 11], we have experimented with using rule-dependency graphs (RDGs) as the explanation structures. Like the CPG and PCPG, an RDG is a directed-acyclic graph that has rule instances as its vertices and directed edges denoting dependencies among the instances. Unlike the former two, in which dependencies are based on chronological precedence, RDG dependencies are strictly based on training example causality.

The SAFE-TO-STACK example's RDG is shown in Figure 11. As with the PCPG, a similar procedure to the one in Figure 9 can be used to compute macrorules from RDGs. Step (1) differs in that a $\texttt{subLDG}_{ij}$ is computed for each $edge(R_i, R_j)$, rather than each $path(R_i, R_j)$, in the RDG. An additional difference is that in step (1a) there *must* be at least one dependency edge between a consequent and an antecedent for each LDG in $\texttt{subLDG}_{ij}$; this is not the case for either the CPG or the PCPG. These differences reduce considerably the total number of macrorules computed from RDGs.

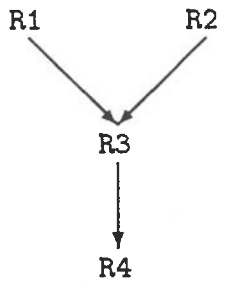

$$\begin{tikzcd} R1 \arrow[dr] & & R2 \arrow[dl] \\ & R3 \arrow[d] & \\ & R4 & \end{tikzcd}$$

Figure 11: Rule-Dependency Graph for the SAFE-TO-STACK Example

Use of the RDG results in only two macrorules, namely Rmacro13 and Rmacro14 in Figure 10, being computed for the SAFE-TO-STACK example. Note that Rmacro14 applies to situations differing structurally from the training example; this compiled knowledge states that an ENDTABLE is SAFE-TO-STACK on an object whose weight, calculated by multiplying its

DENSITY and **VOLUME**, is more than 5, the default weight of an **ENDTABLE**.

The RDG explanation structure provides a way to constrain macrorule generation based on training example causality that is not present in the CPG or PCPG. In the worst case, unfortunately, for n rules there might be $O(n^2)$ edges in the RDG, a situation that would result in having to consider as many dependencies in the first step of Figure 9 as with the CPG. Application dependent experience with Argo, however, reveals that on average the number of edges is much smaller.

Even when the RDG is used, the multiplicative effect on **merged-LDGs** (Figure 9) of having more than one LDG in $\mathtt{subLDG}_{ij}$ per $edge(R_i, R_j)$ can lead to a combinatorial explosion in the number of macrorules computed.[1] Thus, domain-independent schemes for reducing the number of LDGs between pairs of dependent rules in the RDG are desirable. Intuitively, the maximal number of dependencies should be used whenever possible in order to reduce the number of antecedents in the generalization and yield more general macrorules. Because we have assumed rewrite rules, maximal dependencies must be constructed in such a way that at least one consequent/antecedent dependency is guaranteed for each rule dependency in the RDG. Use of maximal dependencies in Argo has proven to be successful in significantly reducing the number of generated macrorules while still preserving operationality with respect to training examples.

One might argue that this maximal-dependency approach could also be used for the CPG and PCPG. Unfortunately, use of maximal dependencies for a given CPG or RDG might lead to a set of macrorules that are not operational with respect to the training example. This situation arises because, unlike dependencies imposed by the RDG, chronological dependencies do not place any causal constraints on rule instances.

7 Abstraction with Rule-Dependency Graphs

Argo, a derivative of the Proteus expert-system development environment [24], is a tool for building knowledge-based systems that transfer experience from previous problem-solving efforts to new problems via analogical reasoning [1, 9, 10, 11]. Problem-solving experience is acquired in the form of a

[1]This situation does not arise in the **SAFE-TO-STACK** example.

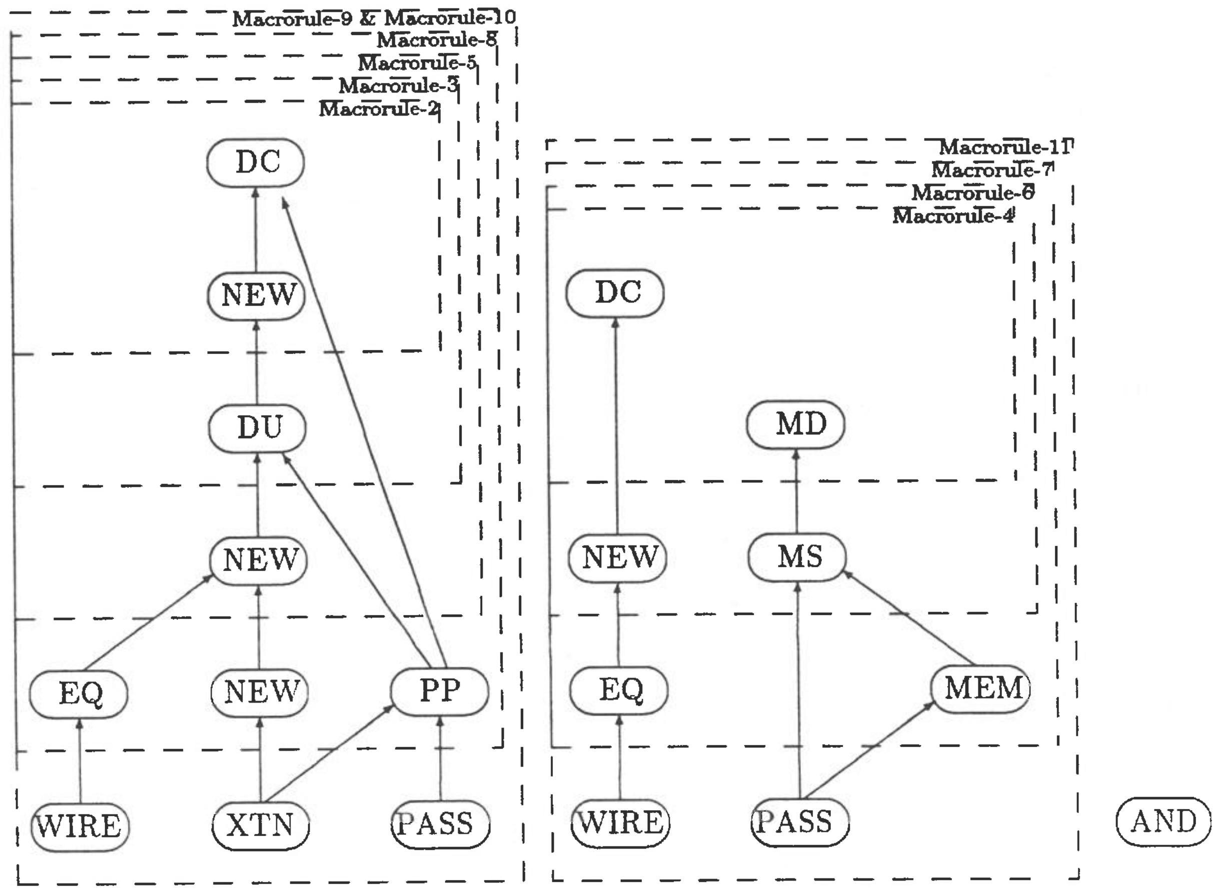
problem-solving plan represented as an RDG. From this graph, Argo calculates a set of macrorules based on increasingly abstract versions of the plan. These macrorules are partially ordered according to an abstraction relation for plans, from which the system can efficiently retrieve the most specific plan applicable for solving a new problem. The use of abstraction in a knowledge-based application of Argo allows the sytem to solve problems that are not necessarily identical, but just analogous to those it has solved previously.

A problem-solving plan in Argo is the explanation structure, in the form of an RDG, for a training example. Argo implements a type of derivational analogy [2, 3, 22] to solve new problems by making use of abstracted RDGs from previous problem-solving experiences. The motivation for using an RDG to represent a problem-solving plan in Argo is that an RDG encompasses a level of detail that facilitates building plan abstractions and learning their corresponding macrorules.

A number of domain-dependent and domain-independent techniques for automatically generating plan abstractions are possible. These include deleting rules from a plan, replacing a rule by a more general rule that refers to fewer details of a problem (as in ABSTRIPS [25]), and generalizing a macrorule for the plan without reference to the plan itself. Argo abstracts a plan by deleting all of its leaf rules, which are those having no outgoing dependency edges. For many domains, the leaf rules trimmed from a plan tend to be those that deal with details at the plan's level of abstraction. Increasingly abstract versions of a plan are obtained by iteratively trimming it until either one or zero nodes remain.

A domain suited for the type of abstractions described above is that of design. In fact, the primary application that has been used for building and testing Argo is a prototype system for VLSI digital circuit design called Argo-V. Figure 12 shows a plan for solving a content-addressable memory (CAM) design problem. This illustration not only shows the RDG, but also how it is abstracted and how many macrorules are computed per abstraction. Note that macrorules are independently computed for each connected subgraph of the plan. Connected subgraphs are associated with independent subproblems within a design and, thus, lead to macrorules whose reusability in future problem solving is improved. For the sake of illustration, alternative circuit designs for portions of the CAM-cell design corresponding to Macrorule-9 and Macrorule-10 appear in Figure 13.

In addition to using RDGs to represent explanation structures and build



DC: Decompose conditional-signal-assignment statements

NEW: Construct new signal-assignment statements from decomposed statements

DU: Decompose unconditional-signal-assignment statements

EQ: Transform a statement containing an equality into a simpler statement

WIRE: Instantiate a connection between two components

PP: Transform a block of signal-assignment statements into ones representing a cascade of pass-transistor networks

XTN: Instantiate an exclusive-OR pass-transistor network

PASS: Instantiate a pass transistor

MD: Decompose an entity into one containing memoried statements and one containing combinational logic

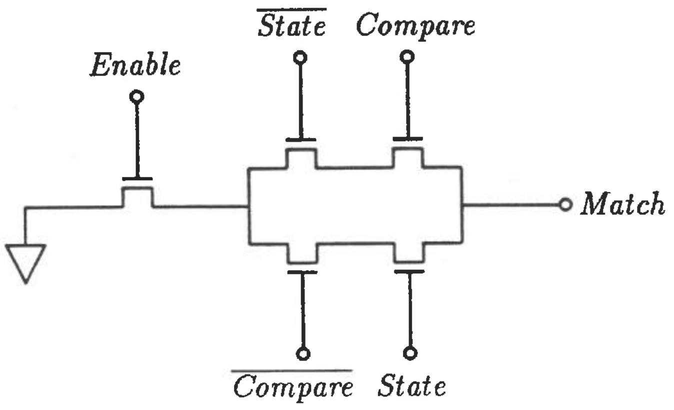
MS: Complete the specification of an entity containing memoried statements

MEM: Instantiate an inverter loop for a one-bit memory

AND: Instantiate an AND gate

Figure 12: Design Plan (RDG) for the CAM-cell Design Problem

Using Macrorule-9:



Using Macrorule-10:

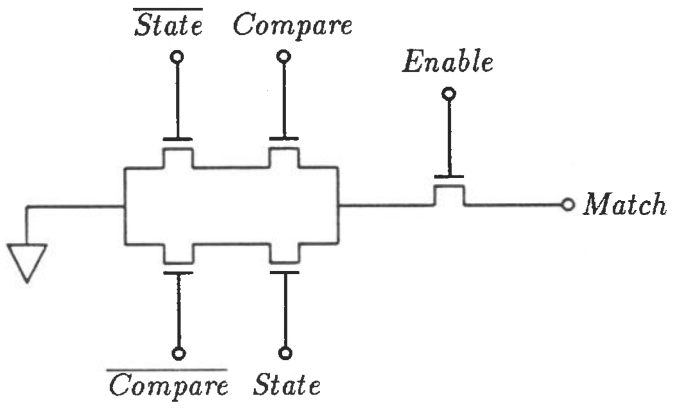


Figure 13: Circuit Designs for Macrorules Learned from a Subproblem of the CAM-cell Training Example

macrorule abstractions, the following extensions to EBL have been implemented in Argo:

- Argo incorporates the Proteus frame system as one of its knowledge representation mechanisms. Frames are organized into an inheritance lattice, thereby enabling multiple inheritance and typed variables. Using Argo's unifier, the types of the variables in macrorules are automatically deduced.

- The use of Lisp is allowed in Argo rules, and thus, Argo's learning mechanism has been designed to learn certain types of macrorules incorporating Lisp. In particular, the learning algorithm accommodates LDGs in which Lisp expressions are bound to variables. This, by the way, is the use of Lisp implied in the `SAFE-TO-STACK` example [20]. A requirement for incorporating a Lisp expression into a macrorule is that the number of times the expression is evaluated must be the same as if the macrorule's component rules had been individually applied.

- Each datum in Argo is included in a justification-based truth-maintenance system (JTMS) and, as such, has a set of justifications and a belief status of IN or OUT [5]. The justification for a macrorule in the JTMS is a list of its component rules. If any of these component rules is invalidated by being given an OUT status, the macrorule is also invalidated. This, in effect, gives Argo a nonmonotonic learning capability [16].

- Argo can compute macrorules from component rules containing specialized second-order predicates, such as `UNLESS`, which implements negation-by-failure, and `ERASE`, which invalidates existing justifications of a datum. The latter enables an emulation of rewrite rules in a manner similar to the "delete" literals in STRIPS operators [6, 7].

8 Conclusions

Most EBL systems compute a macrorule for a training example using the example's proof tree as an explanation structure. Unfortunately, use of proof trees limit the learned knowledge to be structurally equivalent to the training example. This limitation arises from the fine-grained nature of the literal dependencies captured in proof trees. We have presented three alternative explanation structures, based on using rule instances to capture explanations: chronological-precedence graphs, partial chronological-precedence graphs, and rule-dependency graphs. In the Argo system, use of RDGs, which are based on training example causality among rule instances, has proven to be of much benefit in learning abstractions of training examples suitable for analogical reasoning.

Unfortunately, the alternative explanation structures can potentially cause a combinatorial explosion in the number of macrorules computed from a training example. We have suggested the use of maximal edge dependencies for the RDG as a heuristic to reduce, but not preclude, the potential for this combinatorial explosion. Other domain-independent and domain-dependent heuristics can be used incrementally to prune macrorules that are unlikely to be applicable to future problems. These heuristics can be used in conjunction with criteria for establishing the operationality of computed macrorules in order to mitigate the potential combinatorial explosion.

Even if the number of macrorules per training example can be controlled, the number of macrorules learned from many training examples can grow considerably. As presented in [19], this monotonic accumulation of compiled knowledge is a potential flaw of most EBL systems that can eventually lead to a degradation in their performance. Again, heuristics, such as usage counters and operationality criteria, can be used to prune the number of macrorules retained in the system. We are investigating these heuristics, as well as other issues concerning the number of learned macrorules per training example and across training examples.

References

[1] R. D. Acosta, M. N. Huhns, and S. Liuh, "Analogical Reasoning for Digital System Synthesis," *Proceedings of the IEEE International Conference on Computer-Aided Design*, Santa Clara, CA, November 1986, pp. 173–176.

[2] J. G. Carbonell, "Learning by Analogy: Formulating and Generalizing Plans from Past Experience," in *Machine Learning: An Artificial Intel-*

ligence Approach, Vol. I, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, eds., Tioga Press, Palo Alto, CA, 1983, pp. 137–161.

[3] J. G. Carbonell, "Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition," in *Machine Learning: An Artificial Intelligence Approach*, Vol. II, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, eds., Morgan Kaufmann, Los Altos, CA, 1986, pp. 371–392.

[4] G. DeJong and R. Mooney, "Explanation-Based Learning: An Alternative View," *Machine Learning*, vol. 1, no. 2, 1986, pp. 145–176.

[5] J. Doyle, "A Truth Maintenance System," *Artificial Intelligence*, vol. 12, 1979, pp. 231–272.

[6] R. E. Fikes and N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence*, vol. 2, 1971, pp. 189–208.

[7] R. E. Fikes, P. Hart, and N. J. Nilsson, "Learning and Executing Generalized Robot Plans," *Artificial Intelligence*, vol. 3, 1972, pp. 251–288.

[8] C. L. Forgy, *OPS5 User's Manual*, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, July 1981.

[9] M. N. Huhns and R. D. Acosta, "Argo: An Analogical Reasoning System for Solving Design Problems," MCC Technical Report No. AI/CAD-092-87, Microelectronics and Computer Technology Corporation, Austin, TX, April 1987.

[10] M. N. Huhns and R. D. Acosta, "Argo: A System for Design by Analogy," *Proceedings Fourth IEEE Conference on Artificial Intelligence Applications*, San Diego, CA, March 1988.

[11] M. N. Huhns and R. D. Acosta, "Formulating and Retrieving Knowledge through Abstraction Extensions to Explanation-Based Learning," MCC Technical Report No. ACA-AI/CAD-038-88, Microelectronics and Computer Technology Corporation, Austin, TX, January 1988.

[12] S. T. Kedar-Cabelli and L. T. McCarty, "Explanation-Based Generalization as Resolution Theorem Proving," *Proceedings Fourth International Workshop on Machine Learning*, Irvine, CA, June 1987, pp. 383–389.

[13] R. M. Keller, "Defining Operationality for Explanation-Based Learning," *Proceedings AAAI-87*, Seattle, WA, July 1987, pp. 482–487.

[14] R. M. Keller, "Concept Learning in Context," *Proceedings Fourth International Workshop on Machine Learning*, Irvine, CA, June 1987, pp. 91–102.

[15] J. E. Laird, P. S. Rosenbloom, and A. Newell, "Chunking in Soar: The Anatomy of a General Learning Mechanism," *Machine Learning*, vol. 1, no. 1, 1986, pp. 11–46.

[16] S. Liuh and M. N. Huhns, "Using a TMS for EBG," MCC Technical Report No. AI-445-86, Microelectronics and Computer Technology Corporation, Austin, TX, December 1986.

[17] S. Mahadevan, "Verification-Based Learning: A Generalization Strategy for Inferring Problem-Reduction Methods," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August 1985, pp. 616–623.

[18] S. Minton, J. G. Carbonell, "Strategies for Learning Search Control Rules: An Explanation-based Approach," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987.

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

[20] T. M. Mitchell, R. M. Keller, and S. T. Kedar-Cabelli, "Explanation-Based Generalization: A Unifying View," *Machine Learning*, vol. 1, no. 1, 1986, pp. 47–80.

[21] R. J. Mooney and S. W. Bennett, "A Domain Independent Explanation-Based Generalizer," *Proceedings AAAI-86*, Philadelphia, PA, August 1986, pp. 551–555.

[22] J. Mostow, "Automated Replay of Design Plans: Some Issues in Derivational Analogy," to be published in *Artificial Intelligence*, 1988.

[23] N. J. Nilsson, *Principles of Artificial Intelligence*, Tioga Publishing Co., Palo Alto, CA, 1980.

[24] C. J. Petrie, D. M. Russinoff, and D. D. Steiner, "PROTEUS: A Default Reasoning Perspective," *Proceedings of the 5th Generation Computer Conference*, National Institute for Software, Washington, D.C., October 1986.

[25] E. D. Sacerdoti, "Planning in a Hierarchy of Abstraction Spaces," *Artificial Intelligence*, vol. 5, no. 2, 1974, pp. 115–135.

[26] R. Waldinger, "Achieving Several Goals Simultaneously," in *Machine Intelligence 8: Machine Representations of Knowledge*, E. Elcock and D. Michie, eds., Ellis Horwood, Chirchester, UK, 1977, pp. 94-136.