A High-Performance Explanation-Based Learning Algorithm

Alberto Segre
segre@cs.cornell.edu
Department of Computer Science
Cornell University
Ithaca, NY 14853

Charles Elkan
elkan@cs.ucsd.edu
Department of Computer Science and Engineering
University of California at San Diego
La Jolla, CA 92093

Abstract

The main contribution of this paper is a new domain-independent explanation-based learning (EBL) algorithm. The new EBL*DI algorithm significantly outperforms traditional EBL algorithms both by learning in situations where traditional algorithms cannot learn as well as by providing greater problem-solving performance improvement in general. The superiority of the EBL*DI algorithm is demonstrated with experiments in three different application domains. The EBL*DI algorithm is developed using a novel formal framework in which traditional EBL techniques are reconstructed as the structured application of three explanation-transformation operators. We extend this basic framework by introducing two additional operators that, when combined with the first three operators, allow us to prove a completeness result: in the formal framework, every EBL algorithm is equivalent to the application of the five transformation operators according to some control strategy. The EBL*DI algorithm employs all five proof transformation operators guided by five domain-independent control heuristics.

1. Introduction

Explanation-based learning (EBL) [DeJong86, Mitchell86] is a technique for improving the efficiency of problem-solving systems. Intuitively, an EBL algorithm starts with a derivation formed in the course of problem solving (thus *explanation*-based learning), transforms the explanation in order to generalize it, and summarizes the transformed explanation as a new chunk of problem-solving knowledge. While the addition of this new chunk does not in principle change the deductive closure (*i.e.*, the set of solvable problems) of the problem solver, it can have a significant effect on the speed with which those problems are solved; hence EBL is a form of speedup learning [Dietterich86].

The main contribution of this paper is a new domain-independent EBL algorithm that transforms explanations in a more sophisticated way than traditional EBL algorithms. This algorithm, called

From: *Artificial Intelligence* **69**:1-2 (1994)

EBL*DI, significantly outperforms traditional EBL algorithms both by learning in situations where traditional algorithms cannot learn as well as by providing greater speedup in general. This claim is supported via an empirical study in three broadly differing application domains.

The paper is organized as follows. First, Section 2 provides a rational reconstruction of traditional EBL algorithms as the structured application of three general operators for transforming explanations. In Section 3, we describe two intrinsic limitations of traditional EBL algorithms and illustrate these limitations with examples where such algorithms fail to learn useful knowledge. In Section 4, we extend the framework of Section 2 with two additional explanation-transformation operators and establish a completeness result: in the formal framework of Section 2, any EBL algorithm can be reconstructed via some combination of the five explanation-transformation operators. Motivated by the proof of completeness in Section 4, in Section 5 we introduce a new high-performance EBL algorithm that uses five declaratively specified domain-independent heuristics to control the application of all five proof-transformation operators. Finally, Section 6 reports on experiments illustrating the superiority of the EBL*DI algorithm over traditional EBL algorithms in three different domains.

2. A Framework for EBL

Before we can discuss specific EBL algorithms, we must establish the underlying knowledge-representation formalism. For the purposes of this paper, it is reasonable to adopt a very simple formalism: our choice is one involving only *facts* and *rules*. Facts are atomic formulae, or atoms, such as $fragile(chippendale)$ or $expensive(?x)$, where the leading question mark is used to indicate a logic variable.[1] Rules are implications, such as $light(?x) \leftarrow on(?x, ?y) \wedge fragile(?y)$, where the *head* is

[1] Atomic formula, predicate, function, variable, substitution, substitution instance, and other related terms are defined in [Lloyd87]. In addition to the notation used by Lloyd, we will use $\circ$ to denote the ''unify'' relation (*i.e.*, $a \circ b$ *iff* $\exists \theta$ *such that* $a\theta = b\theta$), $\subseteq$ to denote the relation ''is a substitution instance of'' or ''is at least as specific as'' (*i.e.*, $a \subseteq b$ *iff* $\exists \theta$ *such that* $a\theta = b$), $\subset$ to denote the relation ''is a non-trivial substitution instance of'' or ''is strictly more specific than'' (*i.e.*, $a \subset b$ *iff* $a \subseteq b \wedge \neg\, b \subseteq a$), $=$ to denote the relation ''is a variable-renaming substitution instance of'' or ''is exactly as specific/general as'' (*i.e.*, $a = b$ *iff* $a \subseteq b \wedge b \subseteq a$), and $\equiv$ to denote the relation ''is identical to.''

light(?x, ?y) and the *antecedents* are *on*(?x, ?y) and *fragile*(?y). Technically, facts and rules are both first-order definite clauses, with function symbols allowed, but with no special equality predicate. This same formalism underlies most of the work in the logic programming community, in particular the PROLOG programming language, and has also been used to describe most of the previous domain-independent characterizations of EBL [Dietterich90, Hirsh87, Mitchell86, Mooney86].

In this formal framework, a *domain theory* consists of an initial set of facts and rules. The domain theory entails a certain *deductive closure*, which is the collection of all atomic formulae that follow logically from the given domain theory. Problem-solving consists of determining whether or not a given *query*, which may contain some number of existentially quantified variables, is a member of this deductive closure. The query is a member of the deductive closure if an explanation justifying the truth of some substitution instance of the query, *i.e.* a *proof*, can be constructed.

2.1. Explanations as Proofs

For the purposes of this paper, we assume that an EBL algorithm is a function from proofs to rules. Given our knowledge representation formalism of facts and rules, proofs are tree-structured and recursive. Formally:

> **Definition 1**: A *proof* is a tree composed of two types of nodes, *consequent nodes* and *subgoal nodes*, and two types of edges, *rule edges* and *match edges*. Each node n has two tags, a *formula*, denoted $f(n)$, and a *label*, denoted $l(n)$, which are atomic formulae.

A consequent node corresponds to the head of a domain theory rule, while a subgoal node corresponds to an antecedent of a domain theory rule. A match edge links a parent subgoal node to a (unique) child consequent node, while rule edges are used to link a parent consequent node to its child subgoal nodes.

> **Definition 2**: A *consequent node* n_c is a node with zero or more children, denoted $r(n_c)$, connected to n_c via outgoing *rule edges*, and a lone parent, denoted $p(n_c)$.

Definition 3: A *subgoal node* n_s is a node with at most one child, denoted $m(n_s)$, connected to n_s via an outgoing *match edge*, and a lone parent, denoted $p(n_s)$.

The root $root(\rho)$ of a proof ρ is always a subgoal node representing the original query q.

For a given query, problem-solving activity may in general yield zero, one, or more proofs.

Definition 4: A *problem-solving episode* $\Pi_R(q)$ for a given query q and resource bound R yields a series of results $\langle \pi_i \rangle_{i=1}^{i=n}$ such that $n \geq 1$ and

i. $\pi_i = \rho_i$ for $i < n$, where ρ_i is a proof with $l(root(\rho_i)) = q$ and corresponding *answer substitution* $\theta_i = l(root(\rho_i)) \circ f(root(\rho_i))$; and

ii. $\pi_n = fail$, indicating that no further substitution instance of the query q lies within the deductive closure D; or $\pi_n = limit$, indicating that no further substitution instance of the query q lies within the resource-limited deductive closure D_R (although one may well lie within D).

It is sometimes more convenient to refer to the answer substitution series $A_R(q) = \langle \theta_i \rangle_{i=1}^{i=n-1}$ instead of its corresponding problem-solving episode $\Pi_R(q)$.

Next we introduce a notion of soundness for proofs. Informally, a proof is *valid* if it is deductively correct. More formally:

Definition 5: A proof ρ is *valid* if and only if:

i. for each subgoal node with an outgoing match edge $n_s \in \rho$, node formulae are identical across the match edge:

$$f(n_s) = f(m(n_s));$$

ii. for each consequent node $n_c \in \rho$ the logical implication

$$l(n_c) \leftarrow \bigwedge_{n_s \in r(n_c)} l(n_s)$$

follows deductively from the original domain theory; and

iii. for each consequent node $n_c \in \rho$ the logical implication

$$f(n_c) \leftarrow \bigwedge_{n_s \in r(n_c)} f(n_s)$$

is a substitution instance of the logical implication

$$l(n_c) \leftarrow \bigwedge_{n_s \in r(n_c)} l(n_s).$$

The validity of a proof is independent of the original query and the problem-solving system used to construct it: it is an intrinsic property of the proof structure.

It is useful to look at a complete example of a valid explanation. Consider the following simple domain theory consisting of just 9 facts and 3 rules:

$$\begin{array}{lll} k(D) & p(B) & q(?y) \\ k(h(?w)) & n(h(A)) & j(A) \\ p(A) & n(h(B)) & j(C) \end{array}$$

$$\begin{aligned} s(?a) &\leftarrow q(?b) \wedge r(?a, ?b) \\ r(?c, ?d) &\leftarrow p(?e) \wedge m(?c, ?e) \wedge n(?c) \\ m(?f, ?g) &\leftarrow j(?g) \wedge k(?f) \end{aligned}$$

The first result, ρ_1, of the problem solving episode $\Pi(s(?x))$ is shown in Figure 1.[2] The subgoal node $root(\rho_1)$ at the top represents the original query q, with $l(root(\rho_1)) = q = s(?x)$ and $f(root(\rho_1)) = s(h(A))$, a substitution instance of q with answer substitution $\theta = \{?x/h(A)\}$. The consequent node directly below $root(\rho_1)$, $m(root(\rho_1))$, has label $l(m(root(\rho_1))) = s(?a)$, the head of the matching domain theory rule.[3] Each subgoal descendent has its label set to the corresponding rule antecedent, here $q(?b)$ and $r(?a, ?b)$, and its formula set to the appropriately instantiated version of the antecedent, here $q(A)$ and $r(A, A)$. Nodes connected by match edges have identical formulae, while the leaves of the explanation are childless consequent nodes whose labels correspond to domain theory facts and whose formulae correspond to appropriate instances of those facts. Thus it is clear that for any subgoal node n_s, the subtree rooted at n_s provides a valid proof for $f(n_s)$.

2.2. A Generic EBL Algorithm

Perhaps the simplest way to increase the performance of a resource-limited problem solver is to record $f(root(\rho)) = q\theta$ from each proof ρ of each successful problem-solving episode as a new fact in

[2] We omit the subscript R when no resource limit is imposed on the search.

[3] In practice, domain theory rules must be ''standardized apart'' (*i.e.*, a unique variable renaming substitution must be applied to rules at application time) to avoid variable name conflicts between multiple occurrences of the same rule. For clarity, we ensure that variable name conflicts do not occur in the examples of this paper, so the variable names in figures match those in corresponding original domain theory rules.

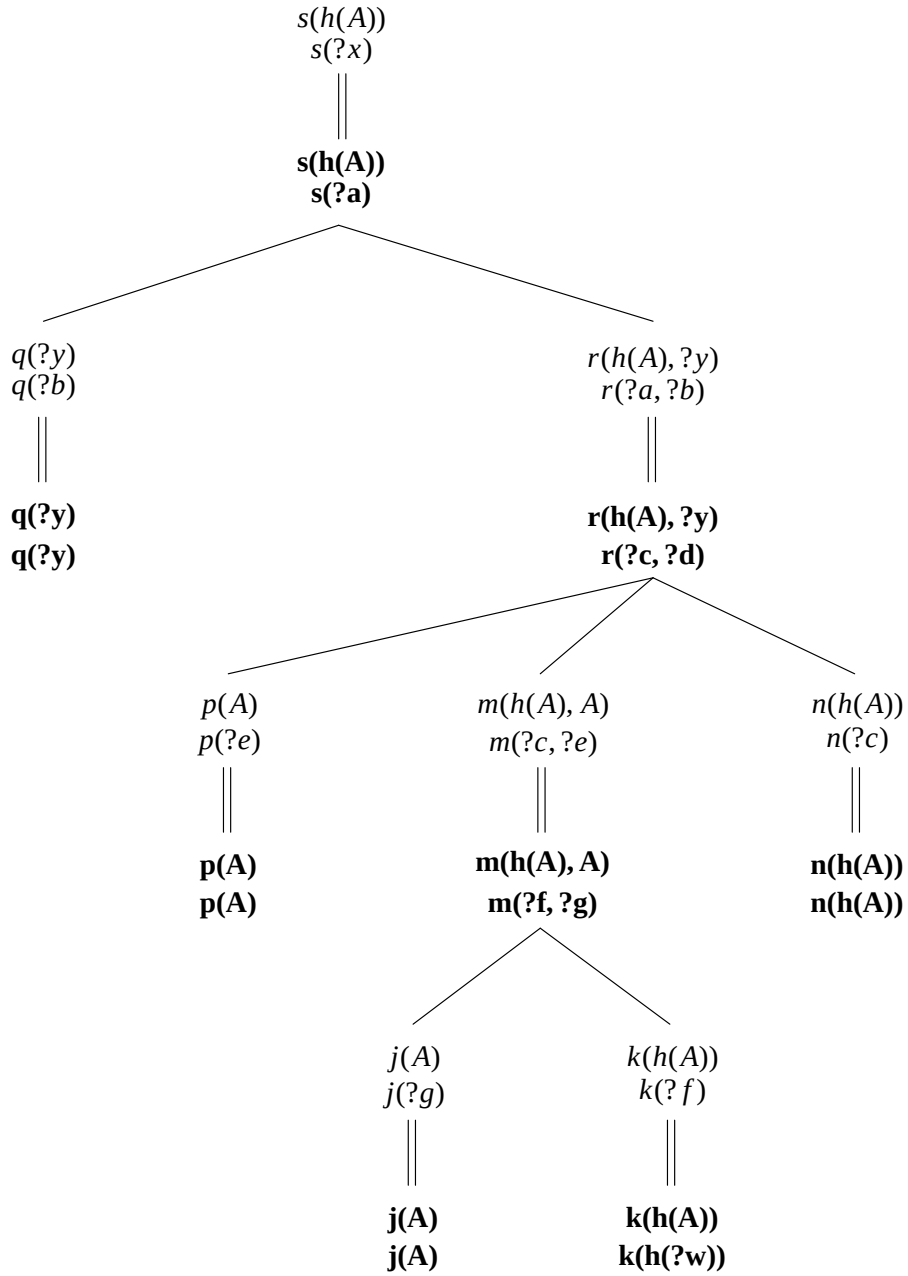

Figure 1: Sample proof. Bold face font is used to indicate consequent nodes, while italic font corresponds to subgoal nodes. The upper expression is the node formula, while the lower expression is the node label. Double lines represent match edges, while rule edges are represented with single lines.

the domain theory. When attempting to prove a similar query again, the prover will immediately find the

new fact and offer this entry as sufficient proof of the query. This rote learning procedure is a restricted form of *success caching*, which, under certain constraints, has already been shown to enhance the performance of this type of prover [Elkan89a, Segre93b].

Unfortunately, rote learning is overly constraining, in the sense that there may exist another form of the query, provable with the same pattern of reasoning implicit in the proof of the current example, which will not match the cached entry. Much more desirable, therefore, is some mechanism by which the chain of logical reasoning used in the proof can be generalized — so as to be more useful — and then retained and reused. This is the essence of EBL: we operate on the structure supporting $root(\rho)$, that is, the subtree rooted at $m(root(\rho))$, in order to generalize it in some validity-preserving manner, and then extract (or *chunk*) a new, more general rule (called a *macro-operator*) to extend the domain theory.

Definition 6: A generic EBL algorithm *generic-ebl* has the form

```
function generic-ebl(ρ: proof): rule;
 begin
  transform(ρ);
  return chunk(ρ);
 end
```

where ρ is the original proof, and *transform*(ρ) leaves the proof ρ in a valid state.

What transformations are required to generalize a proof? Typical transformations performed by EBL algorithms involve pruning away portions of the proof tree. If some of the leaves of a proof tree are unmatched subgoal nodes (*i.e.*, $m(n_s) = \varnothing$) then we say the proof is a *partial proof*. Partial proofs can still be valid; a valid partial proof tree is a demonstration that $f(root(\rho)) = f(m(root(\rho)))$ is implied by the conjunction of its *premises*, or unmatched leaf subgoals. The *chunk* function creates a macro-operator that summarizes the logical argument supporting ρ, thus making this relationship between the premises and $m(\rho)$ explicit.

function *chunk*(ρ: *proof*): *rule*;
 begin
  return $\left\{ f(m(root(\rho))) \leftarrow \bigwedge_{n \in premises(m(root(\rho)))} f(n) \right\}$;

```
 end

function premises(n: node): set of node;
 begin
  if consequent-node?(n) then return ⋃_{s ∈ r(n)} premises(s);
  elseif subgoal-node?(n) ∧ m(n) = ∅ then return {n};
  else return premises(m(n));
 end
```

Of course, if there are no unmatched leaf subgoals in ρ, then $premises(m(root(\rho))) = \varnothing$, and the macro-operator obtained will have no antecedents. Hence rote learning is a special case of *generic-ebl*.

```
function rote(ρ: proof): rule;
 begin
  return chunk(ρ);
 end
```

For the proof of Figure 1, this procedure would produce the new macro-operator

$$s(h(A)) \leftarrow .$$

Logically, this procedure is equivalent to simply adding the new fact $s(h(A)) = f(m(root(\rho))) = f(root(\rho)) = q\theta$ to the domain theory.

2.3. A Reconstruction of Traditional EBL

Given the relationship between EBL and rote learning just described, it is clear that the added power of EBL comes from the proof transformations applied before chunking. Traditional EBL algorithms, such as the EBG [Mitchell86] and EGGS algorithms [Mooney86], generalize explanations by pruning portions of the proof, leaving some number of subgoals unmatched. This effectively relaxes the constraints once imposed by the pruned portions of the proof: once the proof's validity is restored, a macro-operator can be constructed that summarizes the general version of the logical argument used in the original proof. The macro-operator is added to the original domain theory with the provision that, where applicable, it takes precedence over other rules. The addition of the macro-operator will not change the deductive closure of a domain theory, although it may well have a significant effect on the future efficiency of the prover.

A traditional EBL algorithm can be reconstructed as a structured application of the following three basic proof transformation operators:

Definition 7: Operator 1 (*Specialization*). Given a node n and a new expression α that is a substitution instance of the node formula, replace the node formula with the new expression:

$Op1(n, \alpha)$: **if** $\alpha \subseteq f(n)$ **then** $f(n) \Leftarrow \alpha$.

Definition 8: Operator 2 (*Generalization*). Given a node n and a new expression α that is both a substitution instance of the node label and at least as general as the node formula, replace the node formula with the new expression:

$Op2(n, \alpha)$: **if** $f(n) \subseteq \alpha \subseteq l(n)$ **then** $f(n) \Leftarrow \alpha$.

Definition 9: Operator 3 (*Match Edge Pruning*). Given a subgoal node n_s, delete the entire subtree below it:

$Op3(n_s)$: $m(n_s) \Leftarrow \varnothing$.

We can use these three operators to reconstruct a traditional EBL algorithm. As noted earlier, the general idea is to first prune away a portion of the proof, leaving some number of unmatched subgoal nodes, then maximally generalize the proof while still guaranteeing its validity, and finally extract a new macro-operator using the previously introduced function *chunk*:

```
function ebl(ρ: proof): rule;
 begin
  trim(root(ρ));
  lift(root(ρ));
  return chunk(ρ);
 end
```

where *trim* and *lift* together constitute the proof transformation step. Traditional EBL algorithms differ in the exact criteria used to prune the proof (*i.e.*, *trim*) as well as the process used to restore the validity of the proof (*i.e.*, *lift*). The amount of pruning they perform crucially affects the future usefulness of the rule that can be learned from an explanation. This quality of usefulness is traditionally called *operationality* [Braverman88a, Hirsh88, Keller87, Mostow83, Mostow87, Segre87]. It is impossible to determine in isolation whether a new rule will be useful: a formal measure of operationality, in the sense of guaranteeing improved problem-solving performance, has to take into account the distribution of future

queries as well as what other rules are present in the domain theory.

For this reason, EBL systems have in the past relied on various *operationality heuristics* to guide the pruning process. Perhaps the simplest such heuristic is to flag some predicates as operational *a priori*, as in [Mitchell86]. This approach is not always adequate [DeJong86], but it does have the advantage of being explicit. More sophisticated applications of EBL often have more sophisticated notions of operationality. For example, the ARMS system [Segre88, Segre91a] uses syntactic heuristics keyed on the structure of an explanation to determine where to prune. Other operationality heuristics that depend on the semantics of the explanation might also be used; unfortunately, such heuristics are often buried deep within a system, and thus they are often not rendered explicit.

Our reconstruction of traditional EBL relies on a very simple operationality heuristic. Suppose a subgoal is matched to a leaf consequent node, that is, a domain theory fact. It is reasonable to assume that a different version of the subgoal, perhaps with some alternative set of bindings, could also be proven by retrieving the same or a different domain theory fact.

```
procedure trim(n: node);
  begin
    if consequent-node?(n) then for s ∈ r(n) do trim(s);
    elseif subgoal-node?(n) ∧ r(m(n)) = ∅ then Op3(n);
    else trim(m(n));
  end
```

The condition $r(m(n)) = \varnothing$ in the fourth line of procedure *trim* is the same operationality criterion used implicitly in the EGGS algorithm as well as in [Dietterich90]. This procedure simply strips all reference to specific domain theory facts used in the construction of the original proof. Once these axioms are removed, the remaining proof is still valid, since Operator 3 does not affect any of the validity conditions of Definition 5. However, the resulting proof is typically overly constrained, since the binding constraints that were imposed on the proof by the deleted leaf consequent nodes are still implicit in the proof node's formulae. The next step, then, is to ''lift'' the proof, producing a maximally general yet still valid partial

proof structure.

```
procedure lift(n: node);
 begin
  relax-bindings(n);
  apply-bindings(n, collect-bindings(n, ∅));
 end
```

First, we relax all of the binding constraints using Operator 2 by replacing each element of the proof with its corresponding element from the original domain theory, which is available as the node label:

```
procedure relax-bindings(n: node);
 begin
  Op2(n, l(n));
  if consequent-node?(n) then for s ∈ r(n) do relax-bindings(s);
  elseif subgoal-node?(n) ∧ m(n) ≠ ∅ then relax-bindings(m(n));
 end
```

The resulting partial proof no longer contains reference to the constraints implicit in the pruned consequent nodes, but in general it violates the second validity condition of Definition 5. Next, we extract those binding constraints necessary to restore the validity of the proof (*i.e.*, the bindings required to unify the formula and label fields of each node along with the bindings required to enforce unification across proof match edges) and apply them uniformly throughout the entire proof.

```
function collect-bindings(n: node, θ: substitution): substitution;
 begin
  θ ⇐ unify(l(n), f(n), θ);
  if consequent-node?(n) then for s ∈ r(n) do θ ⇐ collect-bindings(s, θ);
  elseif subgoal-node?(n) ∧ m(n) ≠ ∅ then θ ⇐ collect-bindings(m(n), unify(f(n), f(m(n)), θ));
  return θ;
 end
```

Here the function $unify(x, y, \theta)$ returns the substitution θ' that is the result of merging θ with the most general unifier of x and y.

Applying the binding constraints just collected requires a simple recursive descent algorithm that uses Operator 1 to specialize each node.

```
procedure apply-bindings(n: node, θ: substitution);
 begin
  Op1(n, f(n)θ);
  if consequent-node?(n) then for s ∈ r(n) do apply-bindings(s, θ);
  elseif subgoal-node?(n) ∧ m(n) ≠ ∅ then apply-bindings(m(n), θ);
 end
```

Once the process terminates, we have restored the violated validity conditions, ensuring that the resulting proof is once again valid. Figure 2 shows what remains of the sample proof of Figure 1 once the process is complete. We are now ready to extract a new macro-operator using function *chunk*. For the proof in Figure 2, this produces the new macro-operator:

$$s(?a) \leftarrow q(?d) \wedge p(?g) \wedge j(?g) \wedge k(?a) \wedge n(?a)$$

which is considerably more general than the macro-operator obtained by rote learning in Section 2.3.

3. The Limitations of Traditional EBL

Adding the macro-operator produced by an EBL algorithm to the original domain theory should improve performance in subsequent problem-solving episodes. Unfortunately, in practice, applying traditional EBL algorithms in this fashion is subject to two serious problems.

3.1. The Utility Problem

Once a new macro-operator has been added to the domain theory, the hope is that when a future query requires a similar proof structure, this will be found more quickly thanks to the presence of the acquired macro-operator. If the distribution of future problems is favorable, then the prover should exhibit better overall (*i.e.*, faster) performance. It may even solve additional problems which were previously unsolvable within a fixed resource bound. Unfortunately, the effect of EBL may be to slow down the prover. This undesirable effect has been dubbed the *utility problem* [Dietterich86, Minton90].

To see how this can happen, consider the example from the previous section. The acquired macro-operator is intended to accelerate search by providing an alternative, shorter, path to the solution within

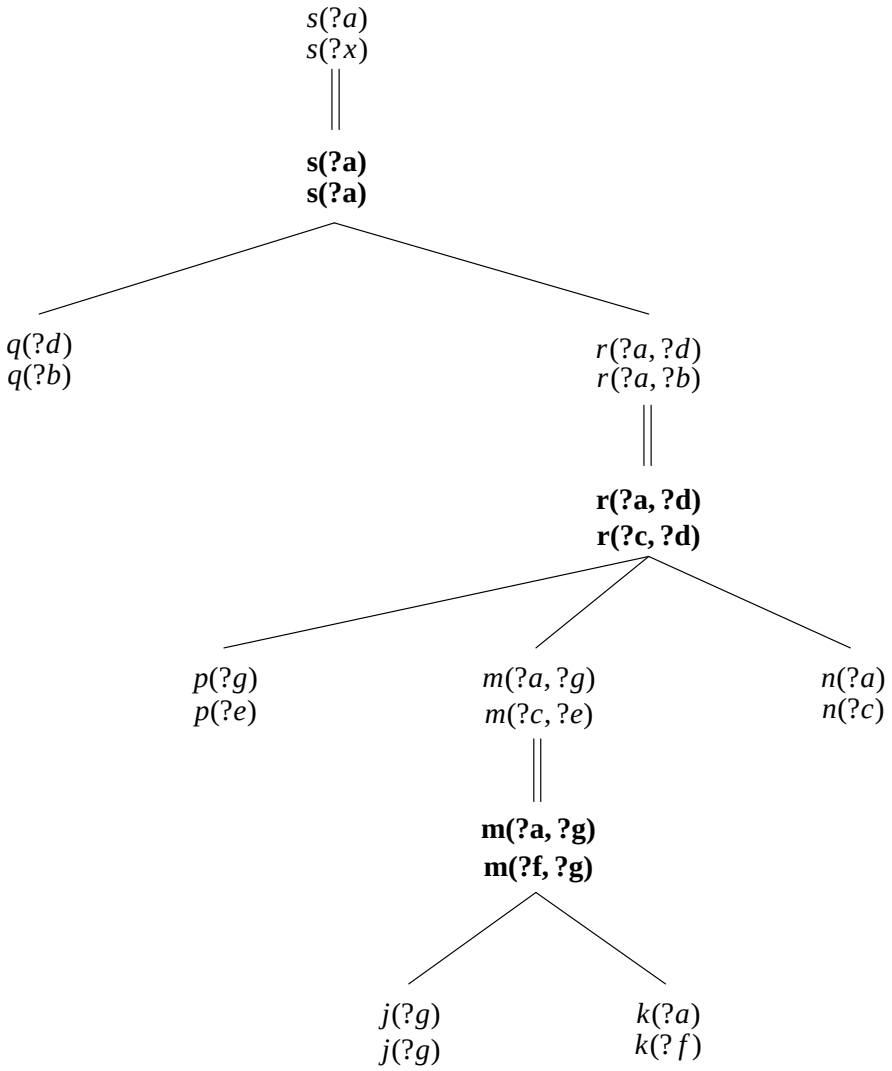

Figure 2: Sample proof of Figure 1 after applying procedure *ebl*. The new macro-operator $s(?a) \leftarrow q(?d) \wedge p(?g) \wedge j(?g) \wedge k(?a) \wedge n(?a)$ is produced by *chunk* from this transformed proof.

the original search space defined by the domain theory. However, if this macro-operator does not lead to a solution for a particular problem, it just defines a redundant path in the search space, and using it causes a region of the search space to be searched again in vain.

While it is impossible to avoid the utility problem altogether, it is possible to minimize its impact. This goal is achieved by reducing both the frequency of inappropriate uses of acquired macro-operators (*i.e.* uses that do not lead to a solution) as well as the cost incurred when an inappropriate use occurs. For example, all else being equal, the overhead of exploring an inappropriate macro-operator typically grows with the number of antecedents in that macro-operator. So for the example above, it is preferable to learn the equally valid (and equally general) macro-operator:

$$s(?a) \leftarrow k(?a) \wedge n(?a).$$

rather than the macro-operator learned by the traditional EBL algorithm, because this macro-operator will entail a smaller performance penalty when used inappropriately thanks to its smaller number of antecedents.

In order to reduce the frequency of inappropriate uses, one might even prefer to learn a more specific version of the macro-operator, such as:

$$s(h(?w)) \leftarrow n(h(?w)).$$

This last macro-operator may not be as useful as the previous one, since its less-general consequent expression means it will be applicable in fewer situations. For this same reason, however, it may less often be used inappropriately. Even when used inappropriately, it will entail a smaller performance penalty, because its single antecedent is a more-specific and therefore easier to prove version of one of the two antecedents of the previous macro-operator.

An example of where the utility problem is serious is the propositional calculus domain of *Principia Mathematica* [Whitehead13] used by Newell, Shaw and Simon in their landmark work on the *Logic Theorist* [Newell63], as adapted for definite-clause theorem provers by [O'Rorke89] and later [Mooney89] (see Appendix B). The traditional EBL algorithm of Section 2.4 performs poorly in the LT domain. In this domain, an EBL algorithm should acquire new macro-operators of a very specific type.

For example, from a proof of $thm(or(P, not(P)))$, we want to learn

$$thm(or(?x, not(?x))) \leftarrow.$$

Given the absence of antecedents, this is essentially a new domain theory fact. We call this type of macro-operator a *generalized cache entry* by virtue of its similarity with success caching (*i.e.*, rote learning), which, for this example, would acquire the strictly more-specific (and therefore less useful) entry:

$$thm(or(P, not(P))) \leftarrow.$$

Generalized cache entries minimize the utility problem: the only overhead in using a generalized cache entry is the added cost of indexing the entry. Yet no existing general-purpose EBL algorithm is capable of recognizing special situations where generalized caching is appropriate.[4]

3.2. Learning from Determinations

A second problem inherent in traditional EBL algorithms can best be introduced by an example. Suppose an artificially intelligent accountant knows that if two stores are located in the same state, then the sales tax rate at both stores must be the same:

$$rate(?y, ?r) \leftarrow state(?x, ?u) \wedge state(?y, ?u) \wedge rate(?x, ?r).$$

Given the common location of *Gucci* and *Cartier* and the sales tax rate at *Gucci*, one can find the rate at *Cartier* as the proof tree in Figure 3 shows.

A useful special-purpose version of the original rule:

$$rate(?x, 7\%) \leftarrow state(?x, NY)$$

states that the sales tax rate at any store in New York is seven percent. This new rule not only follows

[4] The term *generalized caching* is used informally in [Dietterich90] as a synonym for explanation-based learning. Here, by analogy with subgoal caching, we use the term in a more restricted sense to mean only those situations where the acquired macro-operator is a more general version of the original query expression.

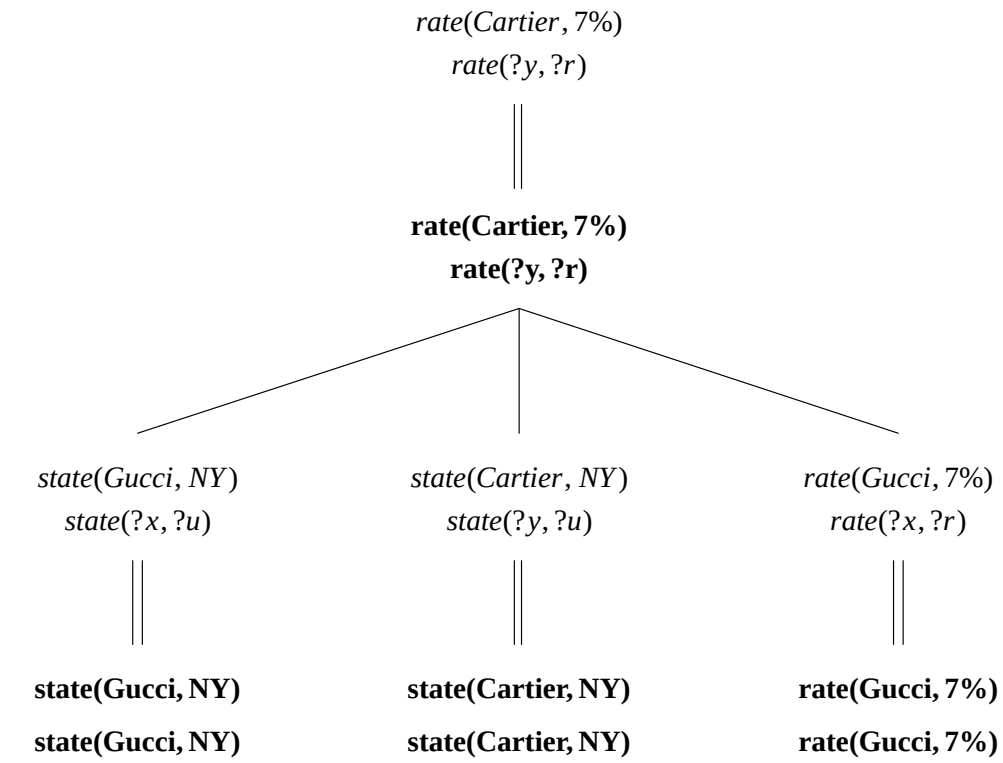

Figure 3: Determining the sales tax rate at Cartier in New York.

deductively from the original domain theory, but is also a useful rule in practice: subsequent queries referring to New York state stores can be handled more efficiently using the new rule than the original domain theory rule. A rule of the form ''stores in the same state pay sales tax at the same rate'' is called a *determination*: a higher-order regularity that by itself is useless in reasoning, but which together with some premises leads to useful conclusions [Davies87]. One can think of a determination as expressing information about similarities between situations in a certain class. Determinations allow the characteristics of a single situation to be extrapolated with confidence.[5]

[5] Determinations have been previously used in EBL work in a quite different way, in order to extend incomplete domain theories. The simplest case of the incomplete domain theory problem occurs when a query is known to be true, yet no proof of the query can be found. When the domain theory gives rise to a single failed proof involving a determination, the PROLEARN-ED algorithm [Mahadevan89] uses the determination to suggest a plausible as-

Traditional EBL algorithms are incapable of acquiring any interesting new macro-operator from a proof involving a determination. If applied to the example proof above, a traditional EBL algorithm yields a macro-operator identical to the original domain theory determination.

4. The EBL* Family of EBL Algorithms

Given the inadequacies of traditional EBL algorithms just described, we would like to extend the framework of Section 2 so that new, more powerful, EBL algorithms (in particular, ones capable of learning from determinations, and ones capable of performing generalized caching under appropriate conditions) can be constructed.

4.1. Two Additional Explanation Transformation Operators

The following two operators complete the EBL* family of proof-transformation operators.

> **Definition 10**: Operator 4 (*Match Edge Grafting*). Given a leaf subgoal node n_s and a consequent node n_c whose formulae unify, graft the proof rooted at n_c at the leaf subgoal n_s:
>
> $$Op4(n_s, n_c) : \textbf{if } m(n_s) = \varnothing \wedge f(n_s) \circ f(n_c) \textbf{ then } m(n_s) \Leftarrow n_c.$$

> **Definition 11**: Operator 5 (*Rule Edge Pruning*). Given a subgoal node n_s and a substitution θ, delete the subtree rooted at n_s while applying the bindings θ to the label of the parent node $p(n_s)$:
>
> $$Op5(n_s, \theta) : r(p(n_s)) \Leftarrow r(p(n_s)) - \{n_s\} \textbf{ and } l(p(n_s)) \Leftarrow l(p(n_s))\theta.$$

Operator 4 can be used (in concert with Operator 3) to emulate the IMEX algorithm [Braverman88b], which ‘‘unravels’’ sections of a proof corresponding to previously acquired macro-operators by suturing in the proof from which the macro-operator was originally derived. Operator 5 enables pruning at rule edges: in particular, when given an appropriate θ, it is this operator that permits us to construct EBL

---

sumption that permits the proof to go through. After asserting this assumption, PROLEARN-ED uses a traditional EBL algorithm to chunk the patched proof. In the special case where the original domain theory is assigned the Clark completion semantics [Clark78], the PROLEARN-ED algorithm is deductively sound. Otherwise, the algorithm is doing a form of abduction, jumping to unsound but plausible conclusions guided by determinations present in the domain theory.

algorithms that can learn the desired macro-operator for the determination example of the previous section.

4.2. Completeness of the EBL* Operators

The five operators described constitute the EBL* repertoire of proof-transformation operators. An EBL* algorithm is an EBL algorithm that transforms proofs by applying EBL* operators in some combination. The EBL* operator repertoire is *complete* in the sense that any macro-operator extracted from a proof by any EBL algorithm can also be learned by an EBL* algorithm.[6]

Theorem: (*Completeness*) Let *ebl* be any EBL algorithm, let ρ be any proof, and let $M = ebl(\rho)$ be the macro-operator extracted from ρ by *ebl*. Then for some EBL* algorithm *ebl**, $M = ebl^*(\rho)$ also.

Proof: Recall from Definition 6 that *generic-ebl* computes M by first applying a validity-preserving proof-modification function *transform* to ρ producing $\bar{\rho}$, and then computing $M = chunk(\bar{\rho})$. Our proof strategy is to show that for any such *transform*, an equivalent function *transform** can be constructed that produces a new proof $\hat{\rho}$ where *transform** uses only EBL* operators and $chunk(\hat{\rho}) = M = chunk(\bar{\rho})$.

By definition, $\bar{\rho}$ is a valid proof, so for each consequent node $\bar{n}_c \in \bar{\rho}$ the logical implication

$$R(\bar{n}_c) = l(\bar{n}_c) \leftarrow \bigwedge_{\bar{n}_s \in r(\bar{n}_c)} l(\bar{n}_s)$$

must follow deductively from the original domain theory.

The first step is to produce an expanded ''canonical'' version of $\bar{\rho}$, denoted $\hat{\rho}$, such that for each consequent node $\hat{n}_c \in \hat{\rho}$ the logical implication

$$R(\hat{n}_c) = l(\hat{n}_c) \leftarrow \bigwedge_{\hat{n}_s \in r(\hat{n}_c)} l(\hat{n}_s)$$

is a substitution instance of a clause in the original domain theory. To construct $\hat{\rho}$, perform a pre-order traversal of $\bar{\rho}$, and, for each consequent node $\bar{v}_c \in \bar{\rho}$ whose corresponding implication is not a substitution instance of a clause in the original domain theory, let $\rho_{\bar{v}_c}$ be a valid partial proof such that $m(root(\rho_{\bar{v}_c}))$ and $\bar{v}_c$ have the same label and formula, and $premises(\rho_{\bar{v}_c}) = r(\bar{v}_c)$, so $chunk(\rho_{\bar{v}_c}) = R(\bar{v}_c)$. Replace each $\bar{v}_c$ node and its children $r(\bar{v}_c)$ by the subtree rooted at $m(root(\rho_{\bar{v}_c}))$, connecting subtrees rooted at each node in $r(\bar{v}_c)$ to their corresponding unmatched leaf subgoal in $\rho_{\bar{v}_c}$.

---

[6] The repertoire is also *minimal* in the sense that none of the operators can be emulated by a combination of the others. The repertoire may, however, not be *minimum*: there may be a set of fewer than five operators which is still complete.

Now $\hat{\rho} = canonicalize(\bar{\rho})$ is a valid proof such that $chunk(\hat{\rho}) = \mathrm{M} = chunk(\bar{\rho})$. The proof is completed by giving a sequence of EBL* operator applications that transforms ρ into $\hat{\rho}$. In brief:

(1) apply Operator 3 to $root(\rho)$;
(2) use Operator 2 to make $f(root(\rho))$ maximally general;
(3) use Operator 4 to construct a proof with the same tree structure as $\hat{\rho}$;
(4) use Operator 5 to remove unwanted subgoals as necessary;
(5) use Operator 1 to specialize the new proof as necessary.

This process can be stated more precisely as a new function *transform**. The essential idea is to use the structure of $\hat{\rho}$ as a template to guide the reconstruction of the desired proof after all but the root of the original derivation has been pruned away.

```
function transform*(ρ: proof): proof;
 var ρ̄: proof ⇐ canonicalize(transform(ρ));
 begin
  Op3(root(ρ));
  Op2(root(ρ), l(root(ρ)));
  dfs-duplicate(root(ρ), root(ρ̄));
  return ρ;
 end

procedure dfs-duplicate(n, n̂: subgoal node);
 begin
  varc: set of node;
  Op4(n, proof-from-clause(clause-from-proof(n̂)));
  c ⇐ r(m(n))
  for s ∈ c do
   if ∃ ŝ ∈ r(m(n̂)) such that l(ŝ) = l(s) then
    begin
     dfs-duplicate(s, ŝ);
     c ⇐ c − s;
    end
  for s ∈ c do Op5(s, unify(l(m(n)), l(m(n̂)), ∅));
 end
```

Here the function *clause-from-proof*, when given $root(\hat{\rho})$, finds the domain theory clause of which $l(m(root(\hat{\rho}))) \leftarrow \bigwedge_{\hat{n} \in r(m(root(\hat{\rho})))} l(\hat{n})$ is an instance. The function *proof-from-clause* makes this clause directly into a corresponding valid partial proof.

□

The proof just given essentially says that the EBL* operations can transform any explanation into the empty explanation and then build up an arbitrary new explanation from scratch.[7] The critical operator that

[7] A similar phenomenon occurs in some planning domains. In a world where there is a ground state attainable from every other state, and the operators for moving from state to state are reversible, then planning is in principle easy: one could go from any state to any other through the ground state.

allows a new proof to be constructed is Operator 4, which can be used to perform a step of backward-chaining inference. The other operators allow the local aspects of a proof to be adjusted. The main technical difficulty in the proof is that the transformed proof $\bar{\rho}$ to be reproduced may not be directly constructible using domain theory clauses. However any consequent/subgoal structure in the proof must itself be an implication provable from the original theory, and a subproof can be notionally spliced in, yielding a proof $\hat{\rho}$ that is easily duplicated by a depth-first traversal.

What is most interesting about the theorem is not the result itself, but rather the insight it gives into the problem of designing EBL algorithms. The difficulty is not in devising explanation transformation methods, rather it is in deciding when it is useful to apply transformations.

5. A Domain-Independent EBL* Algorithm

As the proof of the completeness theorem suggests, the EBL* operators define the space of all alternative partial explanations. Recall the general idea behind EBL is to transform the proof in some fashion, producing a maximally general — yet still valid — partial proof from which a new macro-operator is extracted. EBL algorithms differ in the control heuristics they use to guide the transformation process.

The operationality heuristics used in traditional EBL algorithms (*e.g.*, the *trim* procedure of Section 2.4) are examples of domain-independent control heuristics. While it may be the case that domain-dependent control heuristics are required in order to produce the best possible speedup, here we propose five general heuristics that can be used in constructing domain-independent EBL* algorithms. In particular, these heuristics not only acquire the desired macro-operator in the example of Section 3.2, but also automatically perform generalized caching where appropriate.

The first heuristic, also suggested in [Mooney86], recognizes that chains of reasoning based on single antecedent rules often express taxonomic *isa* relationships, which should not be compiled into the

result of learning lest it become over-specific. Intuitively, the idea is to make the ''highest'' consequent node in a chain look like a domain theory fact.

> **Heuristic 1 (***Trim single-antecedent chains***):** If a leaf subgoal node n_s is an only child, then apply Operator 5 to prune the subtree rooted at n_s while preserving the binding constraints implicit therein:
>
> $H1(n_s)$: **if** $|r(p(n_s))| = 1 \wedge m(n_s)) = \emptyset$
> **then** $Op5(n_s, l(n_s) \circ f(lift(n_s)))$.

The lifting step applied to n_s before its removal ensures that only those binding constraints contributed by the pruned subtree are retained, as opposed to all of the bindings of variables contained in $l(n_s)$.

We may have to apply Heuristic 1 several times to nibble away all single-antecedent structures from the proof. Also note that, since subsequent heuristics may apply Operator 5 (producing consequent nodes with one child that originally had more than one child), it is important to apply Heuristic 1 first so that it is not fooled into identifying such reduced portions of the proof as single antecedent chains. This also ensures that we retain the binding constraints implicit in the pruned subtree, since Heuristic 1 is applied before any significant changes are made to that subtree (*e.g.*, before anything like procedure *trim* is applied). Heuristic 1 is easily implemented as a simple recursive-descent algorithm that eats away at the leaves of the proof.

```
function trim-single-antecedent-chains(n: node): boolean;
 begin
  if subgoal-node?(n) then
   if trim-single-antecedent-chains(m(n)) then
    begin
     lift(n);
     Op5(n, unify(l(n), f(n), ∅));
     return t;
    end
   else return f;
  elseif consequent-node?(n) then
   if r(n) = {s} ∧ trim-single-antecedent-chains(s)) then return t;
   else
    begin
     for s ∈ r(n) do trim-single-antecedent-chains(s);
     return f;
    end
```

end

The second heuristic governs another application of Operator 5. The insight is that certain subgoals provide background information that should be compiled into the learned macro-operator; essentially a form of *partial evaluation* [Van Harmelen88].

> **Heuristic 2 (***Trim alien subgoals***):** If the label $l(n_s)$ of a subgoal node contains only variables not present in the label $l(p(n_s))$ of its parent, then the subtree rooted at n_s should be deleted using Operator 5, while preserving the binding constraints implicit therein.
>
> $H2(n_s)$: **if** $variables(l(p(n_s))) \cap variables(l(n_s)) = \emptyset$
> **then** $Op5(n_s, l(n_s) \circ f(lift(n_s)))$.

The function $variables(x)$ returns the set of variables mentioned in its argument x.

As we shall see in Section 5.1, Heuristic 2 is useful in obtaining the desired generalization for the determination example. Like Heuristic 1, since we are interested in retaining the binding constraints implicit in the pruned subtree, it is important that Heuristic 2 also be applied before anything resembling procedure *trim*. Heuristic 2 is also easily implemented.

```
procedure trim-alien-subgoals(n: node);
 begin
  if consequent-node?(n) then for s ∈ r(n) do trim-alien-subgoals(s);
  elseif subgoal-node?(n) ∧ variables(l(p(n_s))) ∩ variables(l(n_s)) = ∅ then
   begin
    lift(n);
    Op5(n, unify(l(n), f(n), ∅));
   end
  else trim-alien-subgoals(m(n));
 end
```

The third heuristic is a refinement of the traditional EBL operationality heuristic that only removes consequent nodes corresponding to domain theory facts if those nodes actively serve to impose binding constraints on the proof.

> **Heuristic 3 (***Trim axioms selectively***):** If a consequent node n_c is a leaf node whose label and formula are equally specific (*i.e.*, identical subject to variable renaming), apply Operator 3 to prune the subtree rooted at n_c:
>
> $H3(n_c)$: **if** $r(n_c) = \emptyset \wedge f(n_c) = l(n_c)$

then $Op3(p(n_c))$.

Heuristic 3 is more selective than the operationality heuristic of Section 2.4 and plays a critical role in obtaining generalized caching behavior in the LT domain. Its implementation is a straightforward modification of procedure *trim*.

```
procedure trim-axioms-selectively(n: node);
 begin
  if consequent-node?(n) then for s ∈ r(n) do trim-axioms-selectively(s);
  elseif subgoal-node?(n) ∧ r(m(n)) = ∅ ∧ f(n_c) = l(n_c) then Op3(n);
  else trim-axioms-selectively(m(n));
 end
```

The fourth heuristic is deceptively simple to state but, unfortunately, quite expensive to implement.

> **Heuristic 4 (***Trim universally true subproofs***):** If a single answer substitution θ subsumes all other possible answer substitutions for the formula of a subgoal node $f(n_s)$, then the subtree rooted at n_s should be deleted using Operator 5 to preserve θ:
>
> $H4(n_s)$: **if** $\exists\, \theta \in \mathrm{A}(f(n_s))\ \ \forall\, \sigma \in \mathrm{A}(f(n_s))\ \ f(n_s)\sigma \subseteq f(n_s)\theta$
> **then** $Op5(n_s, \theta)$.

Heuristic 4 recognizes that if something can be shown true in the general sense at macro-operator construction time, there is no need to require any verification of the fact at macro-operator application time; simply remove the subgoal in question. It is the problem of recognizing that something is true in the general sense that is so expensive: Heuristic 4 as stated relies on non-resource-bounded proof enumeration to find a substitution that subsumes all other substitutions. We present no implementation of Heuristic 4, however, later, in Section 5.1, we shall see two examples of efficient approximations of Heuristic 4 that can be used to construct practical generalization algorithms.

The last heuristic is perhaps the simplest of all. It recognizes that in some domains there is a certain amount of redundancy in the construction of a proof, such as might occur with frame axioms in situation calculus formulations of planning problems [Green90].

> **Heuristic 5 (***Trim redundant subgoals***):** If two subgoal nodes in a proof have identical formulae (note variable renaming substitutions are *not* allowed), then one of the subgoal nodes should be deleted along with its subtree:

$$H5(n_{s1}, n_{s2}) : \textbf{if } f(n_{s1}) \equiv f(n_{s2}) \textbf{ then } Op5(n_{s2}, \varnothing).$$

Heuristic 5 avoids extracting new macro-operators with redundant antecedents: antecedents that would cause later proofs using the macro-operator to perform the same work more than once. It should only be applied at the end of the proof transformation process, right before the new macro-operator is extracted.

procedure *trim-redundant-subgoals*(*n*: *node*, *a*: *set of atoms*);
 begin
  if *consequent-node?*(*n*) **then for** $s \in r(n)$ **do** *trim-redundant-subgoals*(*s*, *a*);
  elseif *subgoal-node?*(*n*) $\wedge$ $f(n) \in a$ **then** *Op5*(*n*, $\varnothing$);
  else *trim-redundant-subgoals*(*m*(*n*), $a \cup \{f(n)\}$);
 end

5.1. The Control Heuristics at Work

To see how these heuristics might be used, let us return to the tax rate example of Section 3.2. Recall the original proof (Figure 3) is simply an instantiation of the following domain theory rule:

$$rate(?y, ?r) \leftarrow state(?x, ?u) \wedge state(?y, ?u) \wedge rate(?x, ?r).$$

Applying Heuristic 2, we note that none of the variables of the left-most subgoal (*i.e.*, ?*x* and ?*u*) appear in the rule consequent. This subgoal is pruned using Operator 5 and the bindings ?*x*/*Gucci* and ?*u*/*NY* are retained. We next apply Heuristic 3 to remove the two remaining consequent leaf nodes (Operator 3), and use *lift* (Operators 1 and 2) to obtain a maximally general, valid, proof tree. Given that this particular proof is the product of only one rule, the valid proof tree reflects exactly the structure of the original rule, less the pruned subgoal.

The new macro-operator that can be extracted from this partial proof tree is:

$$rate(?y, ?r) \leftarrow state(?y, NY) \wedge rate(Gucci, ?r).$$

While this rule is more useful than the original rule due to the reduction in number of subgoals and variables that must be bound, we can still do better. The rightmost subgoal *rate*(*Gucci*, ?*r*) has an answer substitution {?*r*/7%} that subsumes all other answer substitutions for that subgoal within this domain

theory. By Heuristic 4, it can thus be removed via application of Operator 5 while ‘‘compiling in’’ the substitution {?*r*/7%}:

$$rate(?y, 7\%) \leftarrow state(?y, NY)$$

which is the desired macro-operator.

While the heuristics just described do produce the desired determination, we note that the application of Heuristic 4 in the general case entails enumerating all possible proofs for an expression. Fortunately, two special cases of Heuristic 4 can be implemented efficiently and, when combined, provide much of the power of Heuristic 4.

The first special case of Heuristic 4 recognizes that the null substitution $\varnothing$ subsumes any other substitution. If we can prove a skolemized version of the formula of the node (within some reasonable resource bound), then the formula is universally true, *i.e.,* $\varnothing$ is a valid answer substitution for the original formula.

Heuristic 4a (*Trim universal subproofs***):** If $f(n_s)$ is universally true, then delete the subtree rooted at subgoal n_s:

$H4a(n_s)$: **if** $\varnothing \in \mathrm{A}_R(skolemize(f(n_s)))$
then $Op5(n_s, \varnothing)$.

The function *skolemize*(x) returns a copy of its argument x with each variable replaced by a fresh constant.

Unlike the general statement of Heuristic 4, Heuristic 4a employs resource-bounded search and does not rely on proof enumeration. A reasonable resource bound might be determined by inspecting the resources required to construct the original proof. For example, a resource bound based on the depth of the proof tree rooted at n_s might reasonably be used to limit the depth of search for a universally true equivalent.

```
function trim-universal-subproofs(n: node): boolean;
  var change: boolean ⇐ false;
  begin
```

```
 if consequent-node?(n) then
  begin
    for s ∈ r(n) do change ⇐ change ∨ trim-universal-subproofs(s);
    return change;
  end
 elseif subgoal-node?(n) ∧ prove(skolemize(f(n)), 1 + depth(n)) = ∅ then
  begin
    Op5(n, ∅);
    return t;
  end
 else return trim-universal-subproofs(m(n));
end
```

Heuristic 4a can be used to automatically recognize situations where generalized caching is appropriate, as in the LT domain. An EBL* strategy that includes repeated applications of this approximation to Heuristic 4, in addition to functioning as a normal EBL algorithm, is able to reduce any valid LT proof tree to its maximally general root node; thus recognizing and automatically performing generalized caching as a special case.

The second special case recognizes that if there is only one substitution θ that makes $f(n_s)$ true, then that substitution subsumes all other substitutions.

Heuristic 4b (*Trim singleton subproofs***):** If $f(n_s)$ has only one true substitution, then delete the subtree rooted at subgoal n_s:

$$H4b(n_s): \textbf{if } \Pi_R(f(n_s)) = \langle \rho_1, fail \rangle \wedge \mathrm{A}_R(f(n_s)) = \langle \theta \rangle \textbf{ then } Op5(n_s, \theta).$$

As with Heuristic 4a, a reasonable resource bound might again be determined by examining the original proof structure.

```
function trim-singleton-subproofs(n: node): boolean;
 var change: boolean ⇐ false;
 begin
  if consequent-node?(n) then
   begin
     for s ∈ r(n) do change ⇐ change ∨ trim-singleton-subproofs(s);
     return change;
   end
  elseif subgoal-node?(n) ∧ prove(f(n), 1 + depth(n)) = θ ∧ continue( ) = fail then
   begin
     Op5(n, θ);
```

```
      return t;
    end
  else return trim-singleton-subproofs(m(n));
end
```

It is this second heuristic, which performs a very limited form of resource-bounded proof enumeration, that can be used to prune the *rate*(*Gucci*, ?*r*) subgoal in the tax example. Note, however, that in practice we must temporarily remove the recursive rule for *rate* and perform a resource-limited search only for alternative base cases in order to recognize the proof's singleton nature [Subramanian90].

5.2. The EBL*DI Algorithm

Given the domain-independent heuristics of the previous section, we are now ready to construct a domain-independent EBL* algorithm. Our algorithm, denoted EBL*DI, is easily expressed as a composition of the heuristics introduced previously.

```
function ebl*di(ρ: proof): rule;
 begin
  trim-single-antecedent-chains(root(ρ));
  trim-alien-subgoals(root(ρ));
  trim-axioms-selectively(root(ρ));
  lift(root(ρ));
  while trim-universal-subproofs(root(ρ)) do lift(root(ρ));
  while trim-singleton-subproofs(root(ρ)) do lift(root(ρ));
  trim-redundant-subgoals(root(ρ), ∅);
  return chunk(ρ);
 end
```

Note that some heuristics are applied only once, while others are applied repeatedly as long as they continue to alter the proof. In addition, each heuristic that either violates the validity of the proof or leaves it overly constrained is followed by an application of *lift* to restore the validity and generality of the resulting partial proof.

As with a traditional EBL algorithm, the validity of learned knowledge is dependent on the validity of the original domain theory. Traditional EBL formulations expect the domain theory to be both correct

and stable. Retracting rules or facts from the original domain theory after learning may compromise the validity of any learned rules. Similarly, some EBL* transformation strategies that rely on Heuristic 4 are implicitly dependent on a form of the *closed-world assumption* [Reiter81]. For example, Heuristic 4b is a form of *negation as failure* [Lloyd87]; subsequent addition of a new fact to the domain theory may change the usefulness of the learned rule.

The EBL*DI algorithm is truly a domain-independent learning algorithm in the sense that it is useful over a broad range of domains. Unlike traditional EBL, EBL*DI is not only capable of handling the determination in the tax rate example, but also reduces to performing generalized caching in domains where appropriate, such as the LT domain. Perhaps more interesting, however, is the fact that EBL*DI is free to mix generalized caching on one portion of a proof with the use of a determination in another (or even the same) portion of the proof as appropriate. In Section 6, we support the superior performance of EBL*DI in an empirical comparison with traditional EBL across several domains.

In practice, we expect that improved EBL* transformation strategies for learning macro-operators may well be domain dependent. Taking specific knowledge of a particular domain into account should lead to better, more useful, generalizations. For example, in the tax rate generalization above, ‘‘knowing’’ that a store can have only one tax rate (*i.e.*, that *rate* defines a function from its first argument to its second) would support a very efficient, domain-specific, implementation of Heuristic 4. This kind of information is often easily included in the original domain theory specification: here, for example, as the first-order sentence $\forall\, ?x, ?u, ?v\ \mathit{rate}(?x, ?u) \wedge \mathit{rate}(?x, ?v) \rightarrow ?u = ?v$. Alternatively, if a nonmonotonic semantics such as the standard minimal model semantics is adopted for facts and rules, then functionality assertions are logically entailed, and they can be proven using special inference rules [Elkan88].

6. An Empirical Evaluation of EBL*DI

In this section, we present an empirical comparison of our EBL*DI algorithm with the traditional EBL algorithm of Section 2.4. The central question we want to resolve is whether macro-operators produced our EBL*DI algorithm reliably outperform macro-operators acquired by a traditional EBL algorithm across a spectrum of application domains. We are interested in measuring the change in performance on subsequent problems after learning.

To achieve this goal, typical empirical evaluations compare solution costs (*e.g.*, in terms of elapsed CPU time) for each system over a collection of problems. Such comparisons are at times difficult [Segre91b], but they are a reasonable metric of performance. In this paper, we adopt a simple experimental methodology. More complex model-based methodologies, such as the one used in [Segre93b], can serve to give a clearer picture of system performance. However, for the purpose of the comparisons here, a simple methodology is adequate.

Each experiment reported here uses a different domain theory and problem set. The general idea is to partition the problem set, originally of size N, into two mutually exclusive subsets, a training set of size k and a test set containing the remaining $N - k$ problems. Two otherwise identical depth-first unit-increment iterative-deepening theorem provers are allowed to learn from each problem in the training set using different learning algorithms, and then are tested on the test set.[8] CPU time required and number of nodes explored are recorded for each problem in the test set. These numbers are then compared to those obtained with an identical, non-learning, theorem prover on the same test set. As is normal practice, we assume the cost of learning is negligible in the sense that it can be amortized over many subsequent problems.

[8] A unit increment may well produce the worst-case performance for iterative deepening. Depending on the problem population, increasing the increment value may substantially improve the system's performance.

All of the experiments reported here rely on a heavily instrumented, depth-first iterative-deepening definite-clause theorem prover implemented in Common Lisp. This theorem prover is not particularly fast, since it was designed primarily in order to support principled experimentation. It can be configured to perform iterative deepening [Korf85], iterative broadening [Ginsberg92], conspiratorial best-first iterative deepening [Elkan89a], or even simple breadth-first search. It is the same theorem prover we have used previously for an experimental evaluation of bounded-overhead caching [Segre93b].

6.1. Experiment 1

In this first experiment, we compare the performance of our EBL*DI algorithm and the traditional EBL algorithm in a classic blocks microworld (Appendix A). The problem set consists of 26 problems drawn from a simple situation-calculus formulation of the classic AI block-stacking world [Sussman73]. All of the problems are either singleton goals or binary conjunctions of goals, and no solution requires more than a five move sequence of operators. The relatively small size of the domain is an advantage for this experiment in that it allows us to perform many trials in reasonable amounts of time. For the non-learning system, the largest problem requires searching approximately 20,000 nodes and corresponds to a derivation tree of 84 nodes 7 levels deep.

This blocks-world domain theory is highly recursive and the problem-solving search space is highly redundant. A typical new rule serves only to increase the branching factor of the search space and is of negative utility. In fact, training on a single problem yields useful rules (*i.e.*, produces some overall speedup on the remaining 25 problems) only 8 of 26 times with traditional EBL. The EBL*DI algorithm is superior, in that it produces net speedup for 5 additional problems, for a total of 13 of 26 cases. We used only those 8 problems from which even traditional EBL could extract rules of positive utility to investigate the extent to which the utility of multiple acquired rules is cumulative. For this experiment, our hypotheses are that (*i*) EBL*DI learns macro-operators that give greater speedup than those learned by traditional EBL, and (*ii*) the greater the number of problems from which EBL*DI can learn, the greater

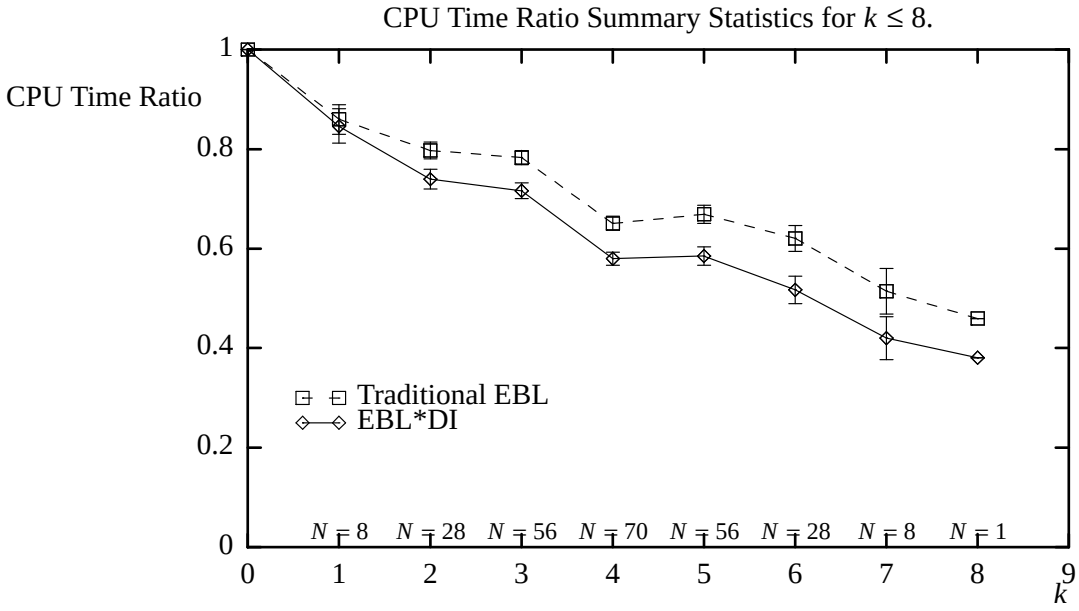

Figure 4: Average CPU time ratios for selected training sets with $k \leq 8$.

the speedup.

The results of experiments testing these hypotheses are illustrated in Figures 4 and 5. The EBL*DI system is significantly faster on average than the EBL system for all tested training sets and training set sizes. Furthermore, learning from more problems makes both systems run faster.

These results are quite impressive: by learning from appropriate problems, the EBL*DI system can solve the remaining test problems can be solved in as little as 32% of the time while searching as few as

[9] It is technically possible for a node exploration ratio under one to correspond to a CPU time ratio over one, since the overhead of using an additional rule may increase the average node exploration cost [Tambe88]. This ‘‘expensive chunk problem’’ does not appear to be an issue in this experiment, as the general shapes of the curves in Figures 4 and 5 are quite similar. More precisely, we observed that average node exploration cost is almost independent of training set size for both learning algorithms. For the traditional EBL system, a very small but statistically significant positive correlation between training set size and average node exploration cost node was observed. However, for the EBL*DI system, we observed a very small but statistically significant *negative* correlation.

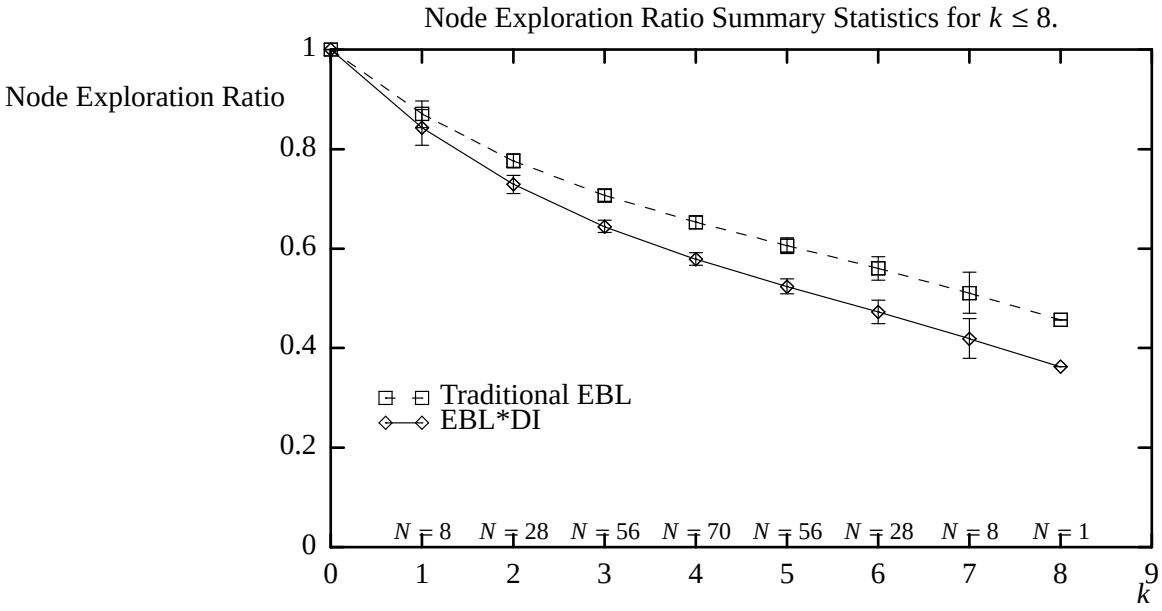

Figure 5: Average node exploration ratios for selected training sets with $k \leq 8$.

33% of the nodes searched by an otherwise equivalent non-learning system. However there are two important caveats. The first is that the results are specific to this particular microworld and problem distribution. We hope that similar results hold in other domains, but no experimental evaluation can prove this.

More important, this experiment finesses the problem of *what* to learn. Learning from random examples is fundamentally more difficult than learning from well-chosen examples supplied by a teacher [Valiant84]. The training sets used in the experiment here consist entirely of problems from which traditional EBL can learn useful macro-operators. Training sets consisting of randomly chosen problems typically do not give speedup in this domain. In the next experiment, we test the performance of EBL*DI in a different domain using teacher-supplied training examples, while in Section 6.3 we show that, in yet another domain, the EBL*DI algorithm achieves speedup when learning from randomly chosen problems.

6.2. Experiment 2

In this experiment, we revisit the classic LT experiment [Newell63], using an updated version of the original LT domain theory (see Appendix B). Queries in the LT domain are statements in the propositional calculus, that is, fully ground expressions, such as $thm(or(not(or(not(P), not(P))), not(P)))$. The 92 propositional calculus problems from Chapter 2 of *Principia Mathematica* are rewritten, replacing *implies* with *or* and *not*: 87 unique problems remain after rewriting. Unlike the blocks microworld problems of the first experiment, the LT problems were originally ordered by the authors of *Principia Mathematica* to maximize their pedagogical utility.

The domain theory consists of three rules and five facts, which correspond to the first five theorems from Chapter 2 of *Principia Mathematica*. Domain theory facts are generalized propositional statements such as $axm(or(not(or(?a, ?a)), ?a))$ that rely on universally quantified variables to allow for constant renaming in the query expressions. In this fashion, both the queries $thm(or(not(or(P, P)), P))$ and $thm(or(not(or(Q, Q)), Q))$ will eventually match the same domain theory fact. The three domain theory rules are

$$\begin{aligned} thm(?x) &\leftarrow axm(?x) \\ thm(?x) &\leftarrow axm(or(not(?y), ?x)) \wedge thm(?y) \\ thm(or(not(?x), ?z)) &\leftarrow axm(or(not(?x), ?y)) \wedge thm(or(not(?y), ?z)). \end{aligned}$$

Since each of these rules has at most one recursive *thm* subgoal, they give rise to the same kind of linear proof structure produced by the original LT system (see Figure 6).

The hypotheses tested in this experiment are that (*i*) learning from early problems helps in solving later problems, and (*ii*) EBL*DI outperforms both traditional EBL and rote learning. The experiment consists of four trials. The first trial is a simple non-learning trial: all 87 problems are attempted, and statistics describing whether or not each problem is solved as well as solution characteristics are recorded. The remaining three trials are learning trials, where a learning algorithm (rote learning, traditional EBL,

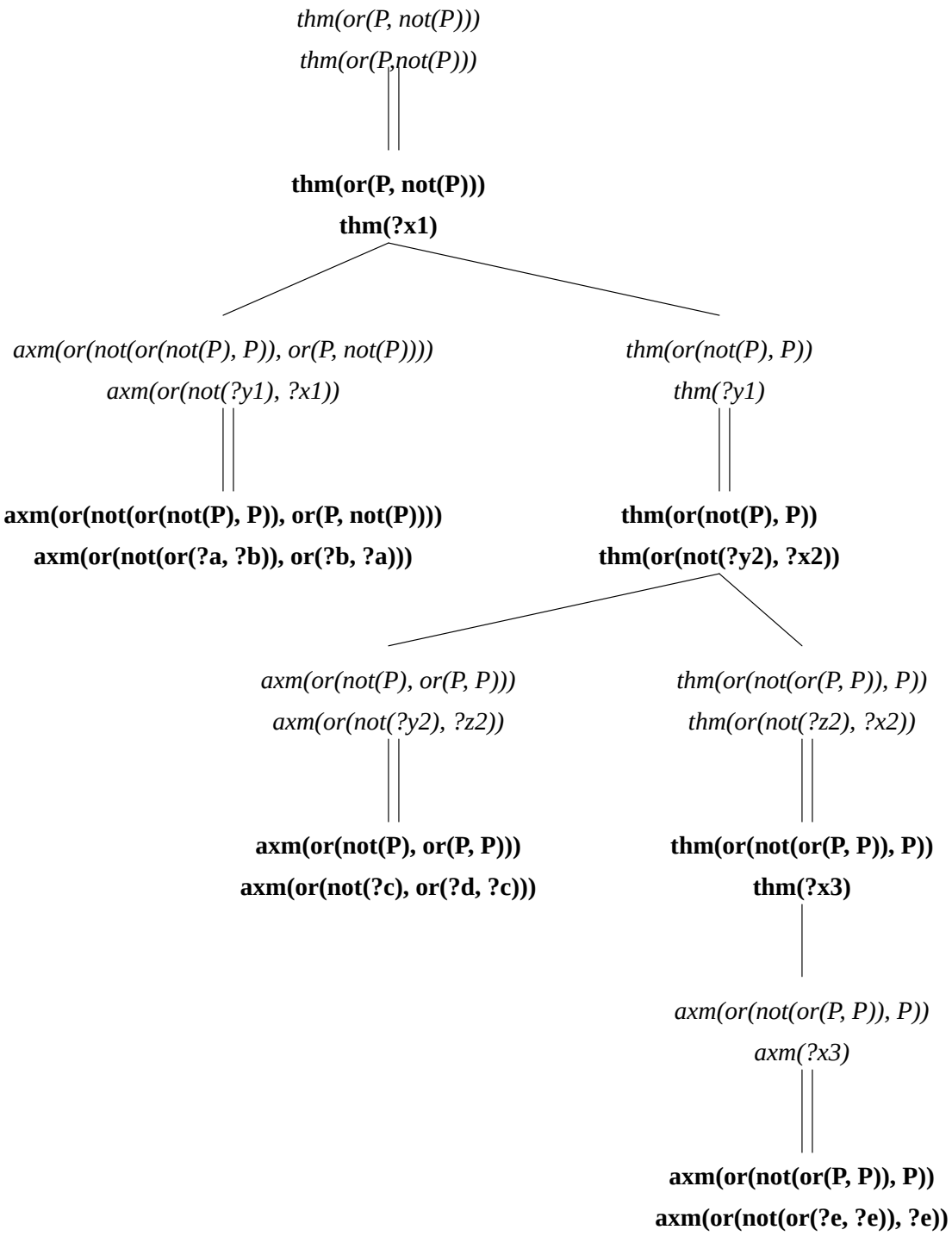

Figure 6: Proof of *thm*(*or*(*P*, *not*(*P*))).

or EBL*DI) is applied to each solved problem with a proof larger than one node; the results of learning are then available for use on subsequent problems.

The protocol just described differs significantly from that used in our first experiment, where separate training and test sets were used. The protocol used here better suits the sequential nature of the LT problem set, and is in the spirit of the original LT experiments.[10] For this reason, we restrict ourselves to qualitative comparisons of the different systems.

Summary statistics for the four trials are shown in Table 1. Each trial was performed under an identical resource bound of 50,000 node explorations per problem. Any problem left unsolved by the non-learning system that was subsequently solved by one of the learning systems was reattempted using the non-learning system with an extended resource bound in order to compute CPU time and node exploration ratios. Such problems are, of course, not included in the results reported for the non-learning system.

Table 1
Summary Results for LT Domain

	No Learning	*Rote*	*EBL*	*EBL*DI*
Problems Solved	34	38	30	44
CPU Time Ratio	1	0.72	16.62	0.13
Node Exploration Ratio	1	0.76	3.13	0.17

[10] The protocol also differs from the original LT protocol of [Newell63], which allowed rote learning (*i.e.*, caching) of unsuccessful problems as new domain theory elements. In general, learning from unproven propositions may augment a domain theory with untrue facts, and it makes the performance contribution of a particular learning algorithm difficult to isolate. Therefore our protocol follows that of [O'Rorke89].

The results are generally in line with those reported in [O'Rorke89]:

(1) The non-learning system solved 34 of the 87 problems within the resource bound. Its CPU time ratio and node exploration ratios are, by definition, 1.

(2) The rote learning system solved 4 additional problems for a total of 38 problems solved. On average, the rote learning system searched fewer nodes (76% of those searched by the non-learning system) and required less time (72% of the CPU time required by the non-learning system).

(3) The traditional EBL system failed to solve 5 problems that were solved by the non-learning system within the resource bound. In return, it was able to solve 1 additional problem not solved by either the non-learning or rote-learning systems. On average, however, the traditional EBL system searched a far greater number of nodes (over 300% of those searched by the non-learning system) and was also much slower than the non-learning system (CPU time ratio of over 1600%) for those problems that it did manage to solve.

(4) The EBL*DI system solved every problem solved by any other tested system. It solved 10 problems more than the non-learning system, 6 problems more than the rote learning system and 14 more than the traditional EBL system. It searched far fewer nodes (17% of those searched by the non-learning system) and was also faster (CPU time ratio of about 13%) than any other system.

These results support our experimental hypotheses. In particular, we see that the traditional EBL algorithm often acquires macro-operators of negative utility. The branching factor of the search explodes as the acquired macro-operators are added to the domain theory. Two factors account for this explosion: the generality of the macro-operator consequents and the number of macro-operator antecedents. On the other hand, the macro-operators acquired by rote learning are nothing more than cached axioms, with very specific consequents and no antecedents. Their specificity guarantees their low overhead, but also makes them less useful.[11] In contrast, since the EBL*DI macro-operators are more general than those acquired by rote learning, they are more widely useful. At the same time, their consequents are not general enough to cause the branching factor explosion observed with traditional EBL.

The principal result is that the EBL*DI algorithm *automatically* performs generalized caching in the LT domain: it is a general-purpose EBL algorithm, and not a special-purpose generalized caching

[11] One of the techniques proposed in [Mooney89] was, in fact, to prohibit chaining on the antecedents of acquired macro-operators in order to reduce the branching factor explosion.

algorithm. In fact, it is precisely the same algorithm used in the experiments of Sections 6.1 and 6.3.

6.3. Experiment 3

One of the original intuitions motivating EBL is that problem solvers are typically posed a series of problems selected according to some skewed yet *a priori* unknown distribution. The desired effect of learning is then to ‘‘tune’’ the problem solver to this particular query distribution, improving its overall expected performance as a consequence. In this experiment, we use a synthetic domain theory to test the performance of traditional EBL and EBL*DI on two different problem distributions, where one of the distributions is uniform and one is weighted to a subspace of problems. Our hypotheses are that (*i*) both EBL algorithms perform better on the skewed distribution, and (*ii*) the EBL*DI algorithm performs better than the traditional EBL algorithm.

In order to have control over the query distribution, we use an artificial theory. The synthetic domain theory used in our experiment (see Appendix C) consists of 330 rules and 38 facts. It was generated in a restricted first-order language without function symbols: thus, while the theory may entail an infinite number of valid proofs, there are only a finite number of atomic formulae within the deductive closure. Restricting the language in this fashion allows us to efficiently compute the deductive closure in a forward-chaining fashion. For the theory used here, there are 976 unique maximally general atomic formulae within the deductive closure.

From this 976 element deductive closure, two 135 element problem sets are generated. The first set is generated from a seed set sampled from the deductive closure according to a uniform probability distribution, while the second set is generated from a seed set drawn according to a skewed probability distribution. Queries are either identical to the seed formula, more specific instances of the seed formula (if it contains universally quantified variables), or copies of the seed formula with new existentially quantified variables replacing randomly selected seed formula constants. Thus while the queries so

generated are guaranteed to have corresponding instances within the domain theory, finding proofs of these queries may involve substantial search (a number of the queries generated do not yield a solution within $2 \cdot 10^7$ node explorations by the non-learning system). The first quasi-uniformly distributed set contains 113 unique queries, while the second set contains 96 unique queries (queries are duplicated due either to repeated seeds or to the introduction of existentially quantified variables in the seed).

Each 135 element problem set is randomly partitioned into an 85 element training set and a 50 element test set. Ten such partitions are generated. For each partition, nine trials are performed for each learning system. The first trial involves learning from 5 randomly selected problems from the training set and testing on all 50 test problems. Each subsequent trial involves training on 10 additional randomly selected training problems. All trials are performed with a 200,000 node exploration resource limit, and

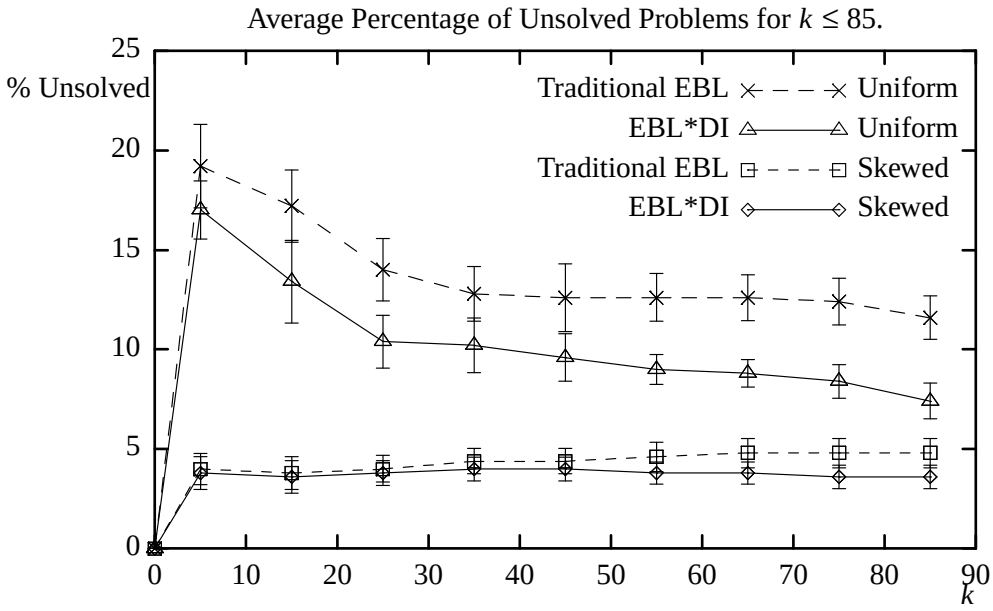

Figure 7: Average percentage of unsolved test problems for nested training sets with $k \leq 85$.

results are compared to the performance of a non-learning system operating on identical test sets.

Figure 7 plots the average percentage of unsolved test problems and makes clear the effect of the utility problem. The non-learning system is able to solve every test problem within the 200,000 node exploration resource limit, while both learning systems lose the ability to solve a certain percentage of the test problems within the same resource bound. The EBL*DI system's intrinsically more useful macro-operators ability to mitigate the adverse effects of the utility problem are especially clear in the quasi-uniform distribution case, where the effects of the utility problem are more pronounced. For the skewed distribution case, the difference is much less striking, especially for smaller training sets. Both learning systems suffer less in the skewed distribution case, where the training problems are by construction more likely to reflect the composition of the test problem set.

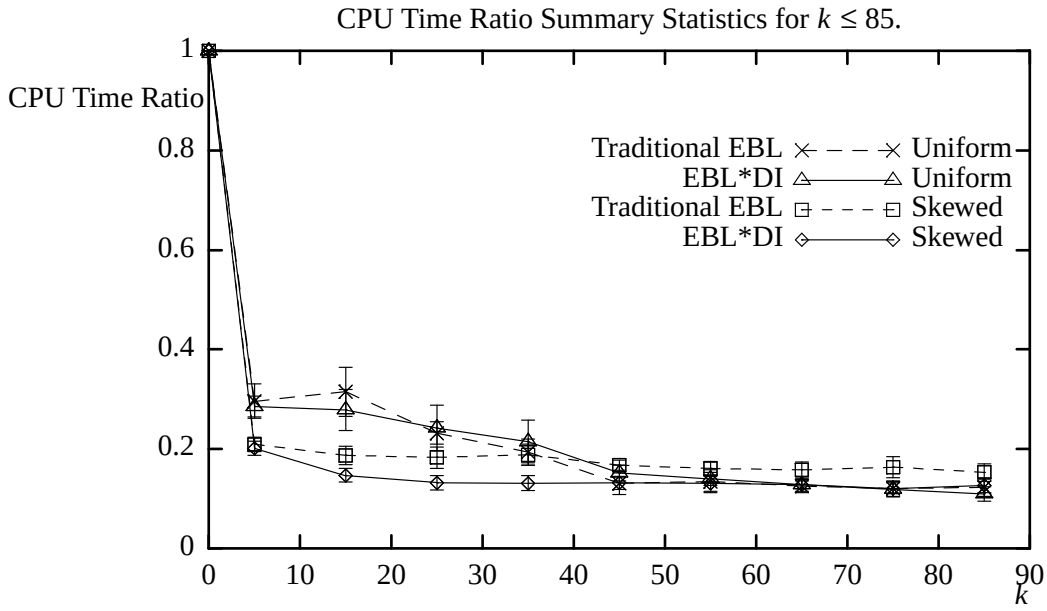

Figure 8: CPU time ratios for nested training sets with $k \leq 85$.

Of course, the adverse effects shown in Figure 7 should be balanced against any performance improvement provided by the macro-operators on the remaining test problems. Figure 8 plots average CPU time ratio for solved problems. As for Figure 7, we again see our hypotheses are supported. First, it is clear that, regardless of learning algorithm employed, the skewed problem distribution case initially yields more CPU time reduction than the quasi-uniform problem distribution case. This is due to the greater ''predictive power'' of the training set with respect to the test set in the skewed distribution case; as the training set size increases, this advantage disappears. Second, we see that the EBL*DI learning system provides greater speedup than the traditional EBL algorithm for both problem distributions. Even where the difference is not large, since the EBL*DI system leaves significantly fewer unsolved problems than the traditional EBL system (Figure 7), this supports our hypothesis that EBL*DI macro-operators are intrinsically more useful than macro-operators produced by a traditional EBL algorithm. Similar conclusions are supported when one examines the node exploration ratios (Figure 9). In summary:

(1) Independent of the distribution tested, the EBL*DI system solves more problems faster and with fewer nodes explored than does the traditional EBL system within the same resource limit.

(2) The performance of both learning algorithms is better on the skewed problem distribution than on the quasi-uniform problem distribution. As expected, the proper selection of training problems has an enormous impact on the overall usefulness of explanation-based learning, regardless of the actual learning algorithm applied.

It should be mentioned that there is one source of experimental bias present in the results just reported. As should be clear from Figure 7, due to the large size of the problems in both problem sets, both systems often did not solve all of the test problems within the resource bounds imposed. For this reason, the results shown in Figures 8 and 9 are *optimistic* estimates of the real values: increasing the resource limits so that more problems are solved would most probably *increase* average CPU time ratios as well as average node exploration ratios. The bias is more pronounced in the quasi-uniform distribution case, where a larger number of problems were left unsolved. Thus, as shown in [Segre91b], it is theoretically possible for the apparent advantage of the EBL*DI system over the EBL system to erode or even to vanish entirely as the resource limit is increased. This eventuality is rather unlikely, however, as

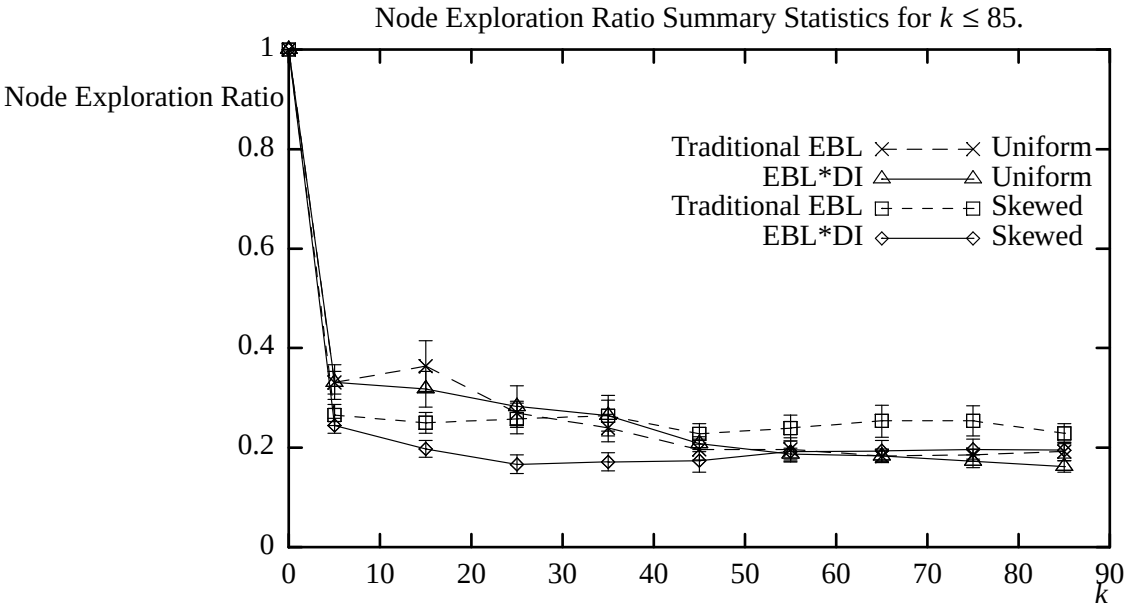

Figure 9: Node exploration ratios for nested training sets with $k \leq 85$.

over all trials and for both distributions the problems solved by EBL*DI and unsolved by traditional EBL outnumber the problems solved by traditional EBL and unsolved by EBL*DI by a margin of greater than 10 to 1.

7. Discussion

The EBL*DI algorithm is superior to traditional EBL algorithms in at least three ways. First, it is able to acquire useful macro-operators in situations where traditional algorithms cannot, such as in the determination example of Section 3.2. Second, it produces macro-operators of significantly greater utility than those produced by traditional EBL algorithms, a claim supported empirically by the experimental results of Section 6. Finally, the generality of EBL*DI's control heuristics — which are declaratively specified — allow the same algorithm to be used across a broad spectrum of application domains, including the LT domain for which a special-purpose EBL algorithm was previously proposed.

The work reported here naturally suggests several topics for further investigation. Perhaps the most important unresolved issue is that added chunks of problem-solving knowledge may still in fact slow down a problem solver. The utility problem is a fundamental dilemma, equally relevant to both forward chaining and backward chaining problem solvers. In the domains we have tested, the EBL*DI algorithm significantly postpones the onset of the utility problem by yielding more useful rules than traditional EBL algorithms.

Research into protocols for using the EBL*DI algorithm could be highly productive, since many protocol issues can influence significantly the utility of macro-operators acquired by EBL, in the case both of control rules as acquired by PRODIGY and of domain rules as acquired by EBL*DI [Markovitch93].[12] The way in which the use of new rules is restricted by the search strategy of the problem-solver is one such issue [Mooney89]. This issue is beyond the scope of the present paper, although the EBL*DI algorithm is designed to perform generalized caching where possible, and generalized caching may be viewed as the combination of general EBL with the search restriction that learned rules must not have subgoals. Generalized caching is also reminiscent of the Soar idea of avoiding expensive chunks by limiting their expressiveness [Tambe90].

Another protocol idea for addressing the utility problem is to merge and generalize control rules inductively [Cohen90]. This idea is not directly compatible with the EBL*DI algorithm, since EBL*DI acquires domain rules and is designed to preserve guaranteed soundness. A third important protocol question is whether learning is done incrementally, or once from a fixed set of training problems. This issue is not fully explored in the present paper, but Experiments 1 and 3 use a separate training phase, while Experiment 2 is incremental, and EBL*DI performs well under both protocols. A final protocol

[12] As mentioned in Section 2.3, caching is a special case of EBL, and some answers to these questions in the context of caching are given in [Segre93b].

question is whether the problems used for learning are randomly selected (as in Experiments 1 and 3), or chosen by a teacher to be especially helpful (as in Experiment 2).

A related issue is the combination of EBL with other speedup learning methods, and in particular with methods that learn from failures. EBL systems that learn control heuristics, as opposed to new domain macro-operators, naturally learn from failed as well as from successful problem-solving episodes. Success and failure caching are known to be cumulative [Elkan89a, Segre92, Segre93a], and integrating EBL from success with EBL from failure has also been suggested [Carpineto90, Pazzani89].

In summary, the main contribution of this paper is the EBL*DI algorithm, a new explanation-based learning method that outperforms traditional EBL algorithms in a diverse set of domains. We have also introduced a novel framework for reconstructing and comparing deductively sound EBL algorithms, and we have specified a family of EBL algorithms that is complete in the sense that it every deductively sound EBL algorithm is equivalent to some algorithm in the family. We have also shown that this framework has practical usefulness, by using it to give a rational reconstruction of a traditional EBL algorithm, as well as a natural specification of the EBL*DI algorithm.

Acknowledgements

Support for this research was provided by the Office of Naval Research through grants N0014-88-K-0123 and N00014-90-J-1542 (AMS), by the Advanced Research Project Agency through Rome Laboratory Contract Number F30602-93-C-0018 via Odyssey Research Associates, Incorporated (AMS), and by the National Science Foundation through grant IRI-9110813 (CPE). The authors also wish to thank two anonymous reviewers for their helpful comments which resulted in a far more precise and better organized paper. This work is based in part on earlier work reported in [Elkan89b].

References

[Braverman88a]
M.S. Braverman and S.J. Russell, "Boundaries of Operationality," *Proceedings of the Fifth International Machine Learning Conference* (June 1988), pp. 221-234.

[Braverman88b]
M.S. Braverman and S.J. Russell, "IMEX: Overcoming Intractability in Explanation Based Learning," *Proceedings of the Seventh National Conference on Artificial Intelligence* (July 1988), pp. 575-579.

[Carpineto90]
C. Carpineto, "Combining EBL from Success and EBL from Failure with Parameter Version Spaces," *Proceedings of the 9th European Conference on Artificial Intelligence*, Pitman (August 1990), pp. 138-140.

[Clark78]
K. Clark, "Negation as Failure," in *Logic and Databases*, H. Gallaire and J. Minker (Eds.), Plenum Press, NY (1978), pp. 293-322.

[Cohen90]
W.W. Cohen, "Learning Approximate Control Rules of High Utility," *Proceedings of the Seventh International Machine Learning Conference* (June 1990), pp. 268-276.

[Davies87]
T.R. Davies and S.J. Russell, "A Logical Approach to Reasoning by Analogy," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* (August 1987), pp. 264-270.

[DeJong86]
G. DeJong and R. Mooney, "Explanation-Based Learning: An Alternative View," *Machine Learning* **1**:2, Kluwer Academic (1986), pp. 145-176.

[Dietterich90]
T. Dietterich, "Machine Learning," *Annual Review of Computer Science* **4**, Annual Reviews, Inc. (1990), pp. 255-306.

[Dietterich86]
T.G. Dietterich, "Learning at the Knowledge Level," *Machine Learning* **1**:3, Kluwer Academic (1986), pp. 287-316.

[Elkan89a]
C. Elkan, "Conspiracy Numbers and Caching for Searching And/Or Trees and Theorem-Proving," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (August 1989), pp. 341-348.

[Elkan88]
C. Elkan and D. McAllester, "Automated Inductive Reasoning about Logic Programs," *Fifth International Conference and Symposium on Logic Programming*, MIT Press (August 1988), pp. 876-892.

[Elkan89b]
C.P. Elkan and A.M. Segre, "Not the Last Word on EBL Algorithms," Technical Report 89-1010, Department of Computer Science, Cornell University, Ithaca, NY (May 1989).

[Ginsberg92]
M.L. Ginsberg and W.D. Harvey, "Iterative Broadening," *Artificial Intelligence* **55**:2-3, North Holland (June 1992), pp. 367-383.

[Green90]
C. Green, "Application of Theorem Proving to Problem Solving," in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate (Eds.), Morgan Kaufmann, San Mateo, CA (1990), pp. 67-87.

[Hirsh87]
H. Hirsh, "Explanation-based Generalization in a Logic-Programming Environment," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* (August 1987), pp. 221-227.

[Hirsh88]
H. Hirsh, "Reasoning About Operationality for Explanation-Based Learning," *Proceedings of the Fifth International Machine Learning Conference* (June 1988), pp. 214-220.

[Keller87]
R.M. Keller, "Defining Operationality for Explanation-based Learning," *Proceedings of the Sixth National Conference on Artificial Intelligence* (July 1987), pp. 482-487.

[Korf85]
R. Korf, "Depth-First Iterative Deepening: An Optimal Admissible Tree Search," *Artificial Intelligence* **27**:1, North Holland (1985), pp. 97-109.

[Lloyd87]
J.W. Lloyd, *Foundations of Logic Programming, Second Edition*, Springer Verlag (1987).

[Mahadevan89]
S. Mahadevan, "Using Determinations in EBL: A Solution to the Incomplete Theory Problem," *Proceedings of the Sixth International Machine Learning Workshop* (June 1989), pp. 320-325.

[Markovitch93]
S. Markovitch and P.D. Scott, "Information Filtering: Selection Mechanisms in Learning Systems," *Machine Learning* **10**:2, Kluwer Academic (February 1993), pp. 113-152.

[Minton90]
S. Minton, "Quantitative Results Concerning the Utility of Explanation-Based Learning," *Artificial Intelligence* **42**:2-3, North Holland (March 1990), pp. 363-392.

[Mitchell86]
T.M. Mitchell, R.M. Keller, and S.T. Kedar-Cabelli, "Explanation-Based Generalization: A Unifying View," *Machine Learning* **1**:1, Kluwer Academic (1986), pp. 47-80.

[Mooney89]
R. Mooney, "The Effect of Rule Use on the Utility of Explanation-Based Learning," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (August 1989), pp. 725-730.

[Mooney86]
R. Mooney and S. Bennett, "A Domain Independent Explanation-Based Generalizer," *Proceedings of the Fifth National Conference on Artificial Intelligence* (August 1986), pp. 551-555.

[Mostow83]
J. Mostow, "A Problem-Solver for Making Advice Operational," *Proceedings of the Third National Conference on Artificial Intelligence* (August 1983), pp. 279-283.

[Mostow87]
J. Mostow, "Searching for Operational Concept Descriptions in BAR, MetaLEX, EBG," *Proceedings of the Fourth International Machine Learning Workshop* (June 1987), pp. 376-382.

[Newell63]
A. Newell, J.C. Shaw, and H. Simon, "Empirical Explorations with the Logic Theory Machine: A Case Study in Heuristics," in *Computers and Thought*, E. Feigenbaum and J. Feldman (Eds.),

McGraw Hill, New York, NY (1963).

[O'Rorke89]
P. O'Rorke, "LT Revisited: Explanation-Based Learning and the Logic of Principia Mathematica," *Machine Learning* **4**:2, Kluwer Academic (November 1989), pp. 117-160.

[Pazzani89]
M. Pazzani, "Explanation-based Learning of Diagnostic Heuristics: A Comparison of Learning from Success and Failure," *Proceedings of the Annual IEEE AI Systems in Government Conference*, IEEE Computer Society (March 1989), pp. 164-168.

[Reiter81]
R. Reiter, "On Closed World Data Bases," in *Readings in Artificial Intelligence*, B.L. Webber and N. Nilsson (Eds.), Morgan Kaufmann, San Mateo, CA (1981), pp. 119-140.

[Segre87]
A.M. Segre, "On The Operationality/Generality Trade-off in Explanation-Based Learning," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* (August 1987), pp. 242-248.

[Segre88]
A.M. Segre, *Machine Learning of Robot Assembly Plans*, Kluwer Academic, Boston, MA (March 1988).

[Segre91a]
A.M. Segre, "Learning How to Plan," *Robotics and Autonomous Systems* **8**:1-2, North Holland (November 1991), pp. 93-111.

[Segre92]
A.M. Segre, "On Combining Multiple Speedup Techniques," *Proceedings of the Ninth International Machine Learning Conference* (July 1992), pp. 400-405.

[Segre91b]
A.M. Segre, C. Elkan, and A. Russell, "Technical Note: A Critical Look at Experimental Evaluations of EBL," *Machine Learning* **6**:2, Kluwer Academic (March 1991), pp. 183-196.

[Segre93a]
A.M. Segre, C.P. Elkan, D. Scharstein, G.J. Gordon, and A. Russell, "Adaptive Inference," in *Foundations of Knowledge Acquisition*, Vol. 2, A. Meyrowitz and S. Chipman (Eds.), Kluwer Academic, Boston, MA (January 1993), pp. 43-81.

[Segre93b]
A.M. Segre and D. Scharstein, "Bounded-Overhead Caching for Definite-Clause Theorem Proving," *Journal of Automated Reasoning* **11**:1, Kluwer Academic (August 1993), pp. 83-113.

[Subramanian90]
D. Subramanian and R. Feldman, "The Utility of EBL in Recursive Domain Theories," *Proceedings of the Eighth National Conference on Artificial Intelligence* (July 1990), pp. 942-951.

[Sussman73]
G.J. Sussman, "A Computational Model of Skill Acquisition," Technical Report 297, MIT Artificial Intelligence Laboratory, Cambridge, MA (1973).

[Tambe88]
M. Tambe and A. Newell, "Some Chunks Are Expensive," *Proceedings of the Fifth International Machine Learning Conference* (June 1988), pp. 451-458.

[Tambe90]
M. Tambe, A. Newell, and P.S. Rosenbloom, "The Problem of Expensive Chunks and its Solution by Restricting Expressiveness," *Machine Learning* **5**:3, Kluwer Academic (August 1990), pp. 299-348.

[Valiant84]
L.G. Valiant, "A Theory of the Learnable," *Communications of the Association for Computing Machinery* **27**:11 (1984), pp. 1134-1142.

[Van Harmelen88]
F. Van Harmelen and A. Bundy, "Explanation-Based Generalisation = Partial Evaluation (Research Note)," *Artificial Intelligence* **36**:3, North Holland (October 1988), pp. 401-412.

[Whitehead13]
A.N. Whitehead and B. Russell, *Principia Mathematica*, Cambridge University Press, Cambridge, England (1913).

Appendix A

Blocks world domain theory and randomly-ordered problem set used for the experiments reported in Section 6.1. The domain theory describes a world containing 4 blocks, *A*, *B*, *C*, and *D*, stacked in various configurations on a *Table*. It consists of 9 facts and 11 rules; there are 26 sample problems.

Facts

$holds(on(A, Table), S_0)$	$holds(on(D, Table), S_0)$	$holds(clear(C), S_0)$
$holds(on(B, Table), S_0)$	$holds(clear(A), S_0)$	$holds(empty(\,), S_0)$
$holds(on(C, D), S_0)$	$holds(clear(B), S_0)$	$holds(clear(Table), ?s)$

Rules

$$holds(and(?x, ?y), ?s) \leftarrow holds(?x, ?s) \wedge holds(?y, ?s)$$
$$differ(?x, ?y) \leftarrow ?x \neq ?y$$

$$holds(holding(?x), do(pickup(?x), ?s)) \leftarrow holds(empty(\,), ?s) \wedge holds(clear(?x), ?s) \wedge differ(?x, table)$$
$$holds(clear(?y), do(pickup(?x), ?s)) \leftarrow holds(on(?x, ?y), ?s) \wedge holds(clear(?x), ?s) \wedge holds(empty(\,), ?s)$$
$$holds(on(?x, ?y), do(pickup(?z), ?s)) \leftarrow holds(on(?x, ?y), ?s) \wedge differ(?x, ?z)$$
$$holds(clear(?x), do(pickup(?z), ?s)) \leftarrow holds(clear(?x), ?s) \wedge differ(?x, ?z)$$

$$holds(empty(\,), do(putdown(?x, ?y), ?s)) \leftarrow holds(holding(?x), ?s) \wedge holds(clear(?y), ?s)$$
$$holds(on(?x, ?y), do(putdown(?x, ?y), ?s)) \leftarrow holds(holding(?x), ?s) \wedge holds(clear(?y), ?s)$$
$$holds(clear(?x), do(putdown(?x, ?y), ?s)) \leftarrow holds(holding(?x), ?s) \wedge holds(clear(?y), ?s)$$
$$holds(on(?x, ?y), do(putdown(?z, ?w), ?s)) \leftarrow holds(on(?x, ?y), ?s)$$
$$holds(clear(?z), do(putdown(?x, ?y), ?s)) \leftarrow holds(clear(?z), ?s) \wedge differ(?z, ?y)$$

Problem Set

$holds(and(on(C, B), on(B, A)), ?s)$	$holds(and(on(C, A), on(D, B)), ?s)$
$holds(on(C, Table), ?s)$	$holds(on(A, D), ?s)$
$holds(and(on(B, D), on(A, C)), ?s)$	$holds(and(on(A, B), clear(D)), ?s)$
$holds(and(clear(D), on(A, B)), ?s)$	$holds(and(on(A, D), on(D, B)), ?s)$
$holds(and(on(D, B), on(B, C)), ?s)$	$holds(and(on(A, C), on(C, B)), ?s)$
$holds(and(on(A, B), on(B, C)), ?s)$	$holds(and(on(B, C), on(A, B)), ?s)$
$holds(on(D, A), ?s)$	$holds(on(A, C), ?s)$
$holds(and(on(D, B), on(C, A)), ?s)$	$holds(on(D, C), ?s)$
$holds(clear(D), ?s)$	$holds(and(on(B, D), on(C, A)), ?s)$
$holds(and(on(C, D), on(A, C)), ?s)$	$holds(and(on(A, C), on(C, D)), ?s)$
$holds(and(on(A, B), on(B, D)), ?s)$	$holds(and(on(A, B), on(D, C)), ?s)$
$holds(and(on(D, A), on(A, C)), ?s)$	$holds(and(on(D, C), on(C, B)), ?s)$
$holds(and(on(B, A), on(C, B)), ?s)$	$holds(and(on(C, B), on(D, C)), ?s)$

Appendix B

Logic Theorist domain theory and ordered problem set used for the experiments reported in Section 6.2. The domain theory consists of 5 facts and 3 rules; there are 87 sample problems.

Facts

$$axm(or(not(or(?a, ?a)), ?a))$$
$$axm(or(not(?a), or(?b, ?a)))$$
$$axm(or(not(or(?a, ?b)), or(?b, ?a)))$$
$$axm(or(not(or(?a, or(?b, ?c))), or(?b, or(?a, ?c))))$$
$$axm(or(not(or(not(?a), ?b)), or(not(or(?c, ?a)), or(?c, ?b))))$$

Rules

$$thm(?x) \leftarrow axm(?x)$$
$$thm(?x) \leftarrow axm(or(not(?y), ?x)) \wedge thm(?y)$$
$$thm(or(not(?x), ?z)) \leftarrow axm(or(not(?x), ?y)) \wedge thm(or(not(?y), ?z))$$

Problem Set

$thm(or(not(or(not(P), not(P))), not(P)))$
$thm(or(not(Q), or(not(P), Q)))$
$thm(or(not(or(not(P), not(Q))), or(not(Q), not(P))))$
$thm(or(not(or(not(P), or(not(Q), R))), or(not(Q), or(not(P), R))))$
$thm(or(not(or(not(Q), R)), or(not(or(not(P), Q)), or(not(P), R))))$
$thm(or(not(or(not(P), Q)), or(not(or(not(Q), R)), or(not(P), R))))$
$thm(or(not(P), or(P, P)))$
$thm(or(not(P), P))$
$thm(or(P, not(P)))$
$thm(or(not(P), not(not(P))))$
$thm(or(P, not(not(not(P)))))$
$thm(or(not(not(not(P))), P))$
$thm(or(not(or(not(not(P)), Q)), or(not(not(Q)), P)))$
$thm(or(not(or(not(P), Q)), or(not(not(Q)), not(P))))$
$thm(or(not(or(not(not(Q)), not(P))), or(not(P), Q)))$
$thm(or(not(or(not(not(P)), P)), P))$
$thm(or(not(P), or(P, Q)))$
$thm(or(not(not(P)), or(not(P), Q)))$
$thm(or(not(P), or(not(not(P)), Q)))$
$thm(or(P, or(not(or(P, Q)), Q)))$
$thm(or(not(P), or(not(or(not(P), Q)), Q)))$
$thm(or(not(or(P, or(Q, R))), or(P, or(R, Q))))$
$thm(or(not(or(P, or(Q, R))), or(or(P, Q), R)))$
$thm(or(not(or(or(P, Q), R)), or(P, or(Q, R))))$
$thm(or(not(or(not(Q), R)), or(not(or(P, Q)), or(R, P))))$

thm(*or*(*not*(*or*(*not*(*Q*), *R*)), *or*(*not*(*or*(*Q*, *P*)), *or*(*P*, *R*))))
thm(*or*(*not*(*or*(*not*(*Q*), *R*)), *or*(*not*(*or*(*Q*, *P*)), *or*(*R*, *P*))))
thm(*or*(*not*(*or*(*P*, *or*(*P*, *Q*))), *or*(*P*, *Q*)))
thm(*or*(*not*(*or*(*Q*, *or*(*P*, *Q*))), *or*(*P*, *Q*)))
thm(*or*(*not*(*or*(*not*(*P*), *or*(*not*(*P*), *Q*))), *or*(*not*(*P*), *Q*)))
thm(*or*(*not*(*not*(*or*(*P*, *Q*))), *not*(*P*)))
thm(*or*(*not*(*not*(*or*(*P*, *Q*))), *not*(*Q*)))
thm(*or*(*not*(*not*(*or*(*P*, *Q*))), *or*(*not*(*P*), *Q*)))
thm(*or*(*not*(*not*(*or*(*P*, *Q*))), *or*(*P*, *not*(*Q*))))
thm(*or*(*not*(*not*(*or*(*P*, *Q*))), *or*(*not*(*P*), *not*(*Q*))))
thm(*or*(*not*(*not*(*or*(*not*(*P*), *Q*))), *or*(*not*(*not*(*P*)), *Q*)))
thm(*or*(*not*(*not*(*or*(*not*(*P*), *Q*))), *or*(*not*(*P*), *not*(*Q*))))
thm(*or*(*not*(*not*(*or*(*not*(*P*), *Q*))), *or*(*not*(*not*(*P*)), *not*(*Q*))))
thm(*or*(*not*(*not*(*or*(*not*(*P*), *Q*))), *or*(*not*(*Q*), *P*)))
thm(*or*(*not*(*or*(*P*, *Q*)), *or*(*not*(*not*(*P*)), *Q*)))
thm(*or*(*not*(*or*(*not*(*not*(*P*)), *Q*)), *or*(*P*, *Q*)))
thm(*or*(*not*(*not*(*P*)), *or*(*not*(*or*(*P*, *Q*)), *Q*)))
thm(*or*(*not*(*not*(*Q*)), *or*(*not*(*or*(*P*, *Q*)), *P*)))
thm(*or*(*not*(*not*(*P*)), *or*(*not*(*Q*), *or*(*not*(*or*(*not*(*P*), *Q*)), *Q*))))
thm(*or*(*not*(*or*(*not*(*P*), *Q*)), *or*(*not*(*or*(*not*(*not*(*P*)), *Q*)), *Q*)))
thm(*or*(*not*(*or*(*P*, *Q*)), *or*(*not*(*or*(*not*(*P*), *Q*)), *Q*)))
thm(*or*(*not*(*or*(*not*(*P*), *Q*)), *or*(*not*(*or*(*P*, *Q*)), *Q*)))
thm(*or*(*not*(*or*(*P*, *Q*)), *or*(*not*(*or*(*P*, *not*(*Q*))), *P*)))
thm(*or*(*not*(*or*(*not*(*P*), *Q*)), *or*(*not*(*or*(*not*(*P*), *not*(*Q*))), *not*(*P*))))
thm(*or*(*not*(*or*(*not*(*or*(*P*, *Q*)), *Q*)), *or*(*not*(*P*), *Q*)))
thm(*or*(*not*(*or*(*not*(*or*(*not*(*P*), *Q*)), *Q*)), *or*(*P*, *Q*)))
thm(*or*(*not*(*or*(*not*(*or*(*not*(*P*), *Q*)), *Q*)), *or*(*not*(*or*(*not*(*Q*), *P*)), *P*)))
thm(*or*(*not*(*or*(*not*(*P*), *Q*)), *or*(*not*(*or*(*or*(*P*, *Q*), *R*)), *or*(*Q*, *R*))))
thm(*or*(*not*(*or*(*not*(*Q*), *P*)), *or*(*not*(*or*(*or*(*P*, *Q*), *R*)), *or*(*P*, *R*))))
thm(*or*(*not*(*or*(*P*, *Q*)), *or*(*not*(*or*(*P*, *or*(*not*(*Q*), *R*))), *or*(*P*, *R*))))
thm(*or*(*not*(*or*(*P*, *or*(*not*(*Q*), *R*))), *or*(*not*(*or*(*P*, *Q*)), *or*(*P*, *R*))))
thm(*or*(*not*(*or*(*not*(*P*), *or*(*not*(*Q*), *R*))), *or*(*not*(*or*(*not*(*P*), *Q*)), *or*(*not*(*P*), *R*))))
thm(*or*(*not*(*or*(*Q*, *R*)), *or*(*not*(*or*(*not*(*R*), *S*)), *or*(*Q*, *S*))))
thm(*or*(*not*(*or*(*not*(*Q*), *or*(*not*(*R*), *S*))), *or*(*not*(*or*(*P*, *Q*)), *or*(*not*(*or*(*P*, *R*)), *or*(*P*, *S*)))))
thm(*or*(*not*(*or*(*or*(*P*, *Q*), *R*)), *or*(*not*(*or*(*or*(*P*, *not*(*R*)), *S*)), *or*(*or*(*P*, *Q*), *S*))))
thm(*or*(*not*(*or*(*not*(*P*), *or*(*not*(*Q*), *R*))), *or*(*not*(*or*(*not*(*P*), *or*(*not*(*R*), *S*))), *or*(*not*(*P*), *or*(*not*(*Q*), *S*)))))
thm(*or*(*not*(*or*(*not*(*or*(*P*, *Q*)), *or*(*P*, *R*))), *or*(*P*, *or*(*not*(*Q*), *R*))))
thm(*or*(*not*(*or*(*not*(*or*(*not*(*P*), *Q*)), *or*(*not*(*P*), *R*))), *or*(*not*(*P*), *or*(*not*(*Q*), *R*))))
thm(*or*(*not*(*and*(*P*, *Q*)), *not*(*or*(*not*(*P*), *not*(*Q*)))))
thm(*or*(*not*(*not*(*or*(*not*(*P*), *not*(*Q*)))), *and*(*P*, *Q*)))
thm(*or*(*not*(*P*), *or*(*not*(*Q*), *and*(*P*, *Q*))))
thm(*or*(*not*(*not*(*and*(*P*, *Q*))), *or*(*not*(*P*), *not*(*Q*))))
thm(*or*(*not*(*or*(*not*(*P*), *not*(*Q*))), *not*(*and*(*P*, *Q*))))
thm(*or*(*not*(*Q*), *or*(*not*(*P*), *and*(*P*, *Q*))))
thm(*or*(*not*(*and*(*P*, *Q*)), *and*(*Q*, *P*)))
thm(*not*(*and*(*P*, *not*(*P*))))
thm(*or*(*not*(*and*(*P*, *Q*)), *P*))

thm(*or*(*not*(*and*(*P*, *Q*)), *Q*))
thm(*or*(*not*(*or*(*not*(*and*(*P*, *Q*)), *R*)), *or*(*not*(*P*), *or*(*not*(*Q*), *R*))))
thm(*or*(*not*(*or*(*not*(*P*), *or*(*not*(*Q*), *R*))), *or*(*not*(*and*(*P*, *Q*)), *R*)))
thm(*or*(*not*(*and*(*or*(*not*(*P*), *Q*), *or*(*not*(*Q*), *R*))), *or*(*not*(*P*), *R*)))
thm(*or*(*not*(*and*(*or*(*not*(*Q*), *R*), *or*(*not*(*P*), *Q*))), *or*(*not*(*P*), *R*)))
thm(*or*(*not*(*and*(*P*, *or*(*not*(*P*), *Q*))), *Q*))
thm(*or*(*not*(*or*(*not*(*and*(*P*, *Q*)), *R*)), *or*(*not*(*and*(*P*, *not*(*R*))), *not*(*Q*))))
thm(*or*(*not*(*and*(*P*, *Q*)), *or*(*not*(*P*), *Q*)))
thm(*or*(*not*(*or*(*not*(*P*), *R*)), *or*(*not*(*and*(*P*, *Q*)), *R*)))
thm(*or*(*not*(*or*(*not*(*Q*), *R*)), *or*(*not*(*and*(*P*, *Q*)), *R*)))
thm(*or*(*not*(*and*(*or*(*not*(*P*), *Q*), *or*(*not*(*P*), *R*))), *or*(*not*(*P*), *and*(*Q*, *R*))))
thm(*or*(*not*(*and*(*or*(*not*(*Q*), *P*), *or*(*not*(*R*), *P*))), *or*(*not*(*or*(*Q*, *R*)), *P*)))
thm(*or*(*not*(*or*(*not*(*P*), *Q*)), *or*(*not*(*and*(*P*, *R*)), *and*(*Q*, *R*))))
thm(*or*(*not*(*and*(*or*(*not*(*P*), *R*), *or*(*not*(*Q*), *S*))), *or*(*not*(*and*(*P*, *Q*)), *and*(*R*, *S*))))
thm(*or*(*not*(*and*(*or*(*not*(*P*), *R*), *or*(*not*(*Q*), *S*))), *or*(*not*(*or*(*P*, *Q*)), *or*(*R*, *S*))))

Appendix C

Synthetic domain theory and problem sets used for the experiments reported in Section 6.3. The domain theory consists of 38 facts and 330 rules; there are two sets of 135 sample problems each.

Facts

$s_0(D)$	$q_0(E, D)$	$n_0(D, E)$	$m_0(E, D, A)$
$s_0(B)$	$q_0(B, B)$	$n_0(D, B)$	$m_0(E, D, E)$
$r_0(B)$	$p_0(B, D)$	$n_0(E, B)$	$m_0(A, ?x, A)$
$r_0(E)$	$p_0(B, ?x)$	$n_0(A, B)$	$m_0(C, B, A)$
$q_0(?x, D)$	$p_0(C, B)$	$m_0(?x, D, ?y)$	$l_0(A)$
$q_0(B, E)$	$p_0(B, C)$	$m_0(A, E, E)$	$l_0(C)$
$q_0(D, D)$	$n_0(D, C)$	$m_0(E, B, C)$	$k_0(E)$
$q_0(D, B)$	$n_0(E, E)$	$m_0(B, B, E)$	$k_0(B)$
$q_0(D, C)$	$n_0(C, D)$	$m_0(D, E, C)$	
$q_0(A, B)$	$n_0(B, A)$	$m_0(B, A, A)$	

Rules

$$
\begin{aligned}
k_1(?i) &\leftarrow n_0(?j, ?i)\\
l_1(?g, ?g) &\leftarrow n_0(?h, ?g)\\
l_1(?c, ?d) &\leftarrow p_0(?e, ?c) \wedge r_0(?f) \wedge m_0(?d, ?c, ?e)\\
l_1(?a, ?a) &\leftarrow k_1(?a) \wedge l_0(?b) \wedge l_1(?b, ?b)\\
m_1(?b, ?c, ?b) &\leftarrow m_0(?c, ?c, ?b)\\
m_1(?j, ?j, ?j) &\leftarrow m_0(?a, ?a, ?j)\\
m_1(?g, ?h, ?g) &\leftarrow p_0(?i, ?h) \wedge r_0(?g)\\
m_1(B, B, B) &\leftarrow l_0(B)\\
m_1(?d, ?d, ?d) &\leftarrow s_0(?e) \wedge r_0(?e) \wedge q_0(?f, ?d)\\
m_1(?b, ?b, C) &\leftarrow n_0(?c, ?c) \wedge l_1(C, C) \wedge p_0(?c, ?b)\\
m_1(?j, ?j, ?a) &\leftarrow k_0(?j) \wedge n_0(?a, ?a)\\
m_1(E, E, E) &\leftarrow r_0(E)\\
m_1(?h, ?h, ?h) &\leftarrow q_0(?i, ?h)\\
m_1(?e, ?e, ?e) &\leftarrow m_0(?f, ?g, ?e)\\
m_1(?b, ?c, ?c) &\leftarrow l_0(?d) \wedge m_0(?c, ?c, ?b)\\
m_1(?h, ?i, ?i) &\leftarrow m_1(?j, ?i, ?h) \wedge m_1(?j, ?a, ?i)\\
m_1(?f, ?f, ?f) &\leftarrow s_0(?f) \wedge q_0(?g, D)\\
m_1(?c, ?c, ?c) &\leftarrow q_0(?d, ?e) \wedge q_0(?d, ?c)\\
m_1(?a, ?b, C) &\leftarrow r_0(C) \wedge s_0(D) \wedge q_0(?b, D) \wedge p_0(?a, ?b)\\
m_1(C, C, C) &\leftarrow l_0(C)\\
m_1(?i, ?j, ?i) &\leftarrow l_0(?i) \wedge k_0(?j)\\
m_1(E, E, E) &\leftarrow s_0(E)\\
m_1(A, A, A) &\leftarrow l_0(A) \wedge s_0(D)\\
m_1(?f, A, ?g) &\leftarrow m_0(A, ?h, A) \wedge q_0(?f, ?g) \wedge m_1(?g, C, ?g)\\
m_1(?c, ?c, ?c) &\leftarrow m_0(?d, ?e, ?c)\\
m_1(?a, ?a, ?a) &\leftarrow l_0(?a) \wedge l_0(?b) \wedge p_0(?b, D)
\end{aligned}
$$

$$
\begin{aligned}
m_1(B,B,B) &\leftarrow q_0(C,D) \wedge l_1(A,B)\\
m_1(?j,?j,?j) &\leftarrow l_0(?j) \wedge k_0(?j) \wedge m_0(?j,?j,?j)\\
m_1(?h,?i,?h) &\leftarrow p_0(?h,?i) \wedge s_0(?h)\\
m_1(E,E,E) &\leftarrow r_0(E)\\
m_1(C,A,C) &\leftarrow r_0(E) \wedge m_0(A,E,C) \wedge r_0(?g) \wedge k_0(E)\\
m_1(?f,?f,?f) &\leftarrow s_0(?f)\\
m_1(?c,?c,?c) &\leftarrow q_0(?d,?d) \wedge m_1(?e,?d,?c)\\
m_1(?a,?b,?b) &\leftarrow k_1(A) \wedge k_1(?b) \wedge q_0(?a,?a)\\
m_1(?i,?j,?i) &\leftarrow r_0(?i) \wedge l_0(?j)\\
n_1(?a,?a,?b) &\leftarrow m_0(B,?b,?a)\\
n_1(?h,?i,?h) &\leftarrow p_0(?j,?h) \wedge l_0(?i) \wedge r_0(?h)\\
n_1(?g,?g,?g) &\leftarrow n_0(?g,?g) \wedge q_0(A,?g)\\
n_1(?e,C,?e) &\leftarrow m_0(?f,?e,C)\\
n_1(?c,E,E) &\leftarrow m_0(?c,?d,E) \wedge k_1(?c)\\
n_1(E,E,?b) &\leftarrow s_0(B) \wedge m_1(B,?b,E)\\
n_1(?h,?h,?h) &\leftarrow m_0(?i,?j,?i) \wedge k_0(?h) \wedge q_0(?a,?j)\\
n_1(?g,?g,?g) &\leftarrow k_1(?g) \wedge p_0(?g,?g)\\
n_1(?d,?e,?d) &\leftarrow p_0(?d,?d) \wedge p_0(?e,?f)\\
n_1(D,D,D) &\leftarrow q_0(D,D)\\
n_1(?a,?a,?a) &\leftarrow m_1(?b,?c,?a) \wedge k_0(?b)\\
n_1(?i,D,?j) &\leftarrow p_0(?j,?j) \wedge r_0(?i) \wedge l_1(?j,D)\\
n_1(?f,?f,?f) &\leftarrow m_0(?g,?h,?h) \wedge m_0(?h,?f,?g) \wedge n_1(?f,?f,?f)\\
n_1(C,C,C) &\leftarrow l_0(C)\\
n_1(?d,?e,?d) &\leftarrow s_0(B) \wedge l_0(?d) \wedge p_0(B,?e)\\
n_1(?b,?b,?b) &\leftarrow m_1(C,?b,?c) \wedge m_0(B,?c,C) \wedge n_1(?c,?b,?c)\\
n_1(?i,?i,?i) &\leftarrow m_0(?j,?j,?j) \wedge k_1(?i) \wedge s_0(?i) \wedge p_0(?a,?j)\\
n_1(A,?h,B) &\leftarrow p_0(?h,D) \wedge p_0(A,B)\\
n_1(?e,?f,?f) &\leftarrow l_0(?g) \wedge l_1(?g,?e) \wedge n_1(?e,?f,?e)\\
n_1(D,E,E) &\leftarrow p_0(D,D) \wedge n_1(E,E,E) \wedge r_0(B)\\
n_1(A,A,A) &\leftarrow l_0(A) \wedge r_0(A)\\
n_1(?d,?d,?d) &\leftarrow r_0(?d)\\
n_1(?b,?b,?b) &\leftarrow l_1(?c,?b) \wedge n_0(?c,?b)\\
n_1(?h,?h,?i) &\leftarrow m_0(?j,?a,?a) \wedge m_0(?i,?j,?h)\\
n_1(D,D,B) &\leftarrow q_0(B,E) \wedge m_1(D,E,E) \wedge k_0(B)\\
n_1(?g,?g,?g) &\leftarrow k_0(?g) \wedge s_0(?g)\\
n_1(?d,?d,?d) &\leftarrow m_0(?e,?e,?f) \wedge n_1(?e,?d,?e)\\
p_1(?d,?d,?e) &\leftarrow n_0(D,?d) \wedge k_0(?e)\\
p_1(?a,?a,?a) &\leftarrow m_0(?b,?c,B) \wedge l_0(?a)\\
p_1(?i,?i,?i) &\leftarrow l_1(?j,?j) \wedge p_0(?i,?j) \wedge n_0(?j,?j)\\
p_1(?g,?g,?g) &\leftarrow n_0(?h,?g)\\
p_1(?e,?e,?e) &\leftarrow q_0(?f,?e)\\
p_1(?d,?d,?d) &\leftarrow k_0(?d)\\
p_1(?b,?b,?c) &\leftarrow p_0(?c,?b)\\
p_1(C,C,C) &\leftarrow p_0(A,C)\\
p_1(?h,?i,?h) &\leftarrow l_0(?j) \wedge p_1(?i,?a,?h) \wedge s_0(B)\\
p_1(?f,?f,?f) &\leftarrow s_0(?g) \wedge s_0(?f)\\
p_1(?d,?d,?d) &\leftarrow n_0(E,B) \wedge k_0(B) \wedge k_0(?d) \wedge k_1(?e)
\end{aligned}
$$

$$
\begin{aligned}
p_1(?b, ?b, ?c) &\leftarrow p_0(?c, ?b) \wedge r_0(?b)\\
p_1(A, A, A) &\leftarrow p_0(B, A)\\
p_1(B, B, B) &\leftarrow p_1(B, B, B) \wedge s_0(D)\\
p_1(C, E, B) &\leftarrow m_0(B, C, E)\\
p_1(D, D, B) &\leftarrow p_0(D, B) \wedge m_0(E, A, A)\\
p_1(?a, ?a, ?a) &\leftarrow k_0(E) \wedge k_1(?a) \wedge l_0(C)\\
p_1(?g, ?g, ?g) &\leftarrow n_0(?h, ?h) \wedge l_0(?i) \wedge n_1(?h, ?j, ?g)\\
p_1(?b, ?b, ?b) &\leftarrow m_1(?c, ?d, ?b) \wedge q_0(?d, ?e) \wedge l_0(?f)\\
p_1(?h, ?i, ?j) &\leftarrow m_0(?j, ?h, ?a) \wedge p_1(?j, ?h, ?a)\\
p_1(?f, B, ?g) &\leftarrow m_1(?f, ?g, B) \wedge k_0(?g)\\
p_1(?d, ?d, ?d) &\leftarrow m_0(B, ?e, B) \wedge l_1(?d, B)\\
p_1(?b, ?b, ?b) &\leftarrow p_0(?c, ?b)\\
p_1(?i, ?i, ?i) &\leftarrow l_0(?i) \wedge m_0(?j, ?a, ?i)\\
p_1(A, B, A) &\leftarrow r_0(B) \wedge p_1(A, A, A)\\
p_1(A, A, A) &\leftarrow l_0(A)\\
p_1(D, D, ?h) &\leftarrow s_0(?h) \wedge n_1(C, D, ?h) \wedge r_0(D) \wedge n_0(C, ?h)\\
p_1(E, E, E) &\leftarrow r_0(E) \wedge k_0(E)\\
p_1(?c, ?c, ?c) &\leftarrow q_0(?d, ?e) \wedge k_1(?f) \wedge n_1(?d, ?c, ?g)\\
q_1(?j, ?a, ?j) &\leftarrow n_0(?b, ?a) \wedge p_0(?c, ?j)\\
q_1(?h, ?h, ?h) &\leftarrow q_0(?i, ?h)\\
q_1(B, E, E) &\leftarrow s_0(E) \wedge k_0(B) \wedge l_0(C)\\
q_1(?f, ?g, ?g) &\leftarrow p_0(?g, ?f)\\
q_1(?b, ?b, ?b) &\leftarrow n_1(?c, ?d, ?d) \wedge p_0(?c, ?e) \wedge m_1(?b, ?d, ?c) \wedge q_1(?e, ?c, ?d)\\
q_1(?a, ?a, ?a) &\leftarrow s_0(?a)\\
q_1(?h, ?h, ?h) &\leftarrow s_0(?h) \wedge m_0(?i, ?i, ?j)\\
q_1(?e, ?f, ?f) &\leftarrow k_0(?g) \wedge l_0(?e) \wedge q_1(?f, ?f, ?g)\\
q_1(?c, ?c, ?c) &\leftarrow n_0(?d, ?c)\\
q_1(?b, ?b, ?b) &\leftarrow k_1(?b) \wedge q_0(?b, B) \wedge p_1(B, B, ?b)\\
q_1(?j, ?j, ?j) &\leftarrow k_0(?j) \wedge l_0(?a)\\
q_1(?i, ?i, ?i) &\leftarrow m_0(?i, C, B)\\
q_1(?e, ?f, ?e) &\leftarrow l_0(?e) \wedge r_0(?g) \wedge p_0(?h, ?e) \wedge q_0(?f, ?f)\\
q_1(?c, ?c, ?d) &\leftarrow s_0(?c) \wedge p_0(?d, D)\\
q_1(?b, ?b, ?b) &\leftarrow s_0(?b)\\
q_1(E, ?a, ?a) &\leftarrow m_0(?a, D, ?a) \wedge m_0(E, D, ?a)\\
q_1(?h, ?h, ?h) &\leftarrow p_0(?i, ?j) \wedge p_1(?h, B, B) \wedge q_0(B, B)\\
q_1(?e, ?e, ?f) &\leftarrow p_0(?g, ?g) \wedge q_0(?f, ?e) \wedge k_1(?e)\\
q_1(?b, ?b, ?b) &\leftarrow m_0(?c, ?d, ?b)\\
q_1(D, D, C) &\leftarrow m_0(C, B, A) \wedge m_1(C, D, A)\\
q_1(?a, ?a, ?a) &\leftarrow k_1(?a) \wedge s_0(B)\\
q_1(?h, ?h, ?i) &\leftarrow r_0(?j) \wedge m_1(?h, ?i, ?i)\\
q_1(?f, ?f, ?f) &\leftarrow m_0(?f, ?f, ?g) \wedge k_0(?g)\\
q_1(B, B, B) &\leftarrow l_0(B)\\
q_1(?e, ?e, ?e) &\leftarrow r_0(?e)\\
q_1(D, D, D) &\leftarrow k_0(E) \wedge s_0(D)\\
q_1(?c, ?c, ?c) &\leftarrow p_0(B, D) \wedge s_0(B) \wedge n_1(?d, D, ?c)\\
q_1(?b, B, B) &\leftarrow s_0(?b) \wedge s_0(B)\\
q_1(B, B, B) &\leftarrow r_0(B)
\end{aligned}
$$

$$
\begin{aligned}
q_1(?i,?i,?i) &\leftarrow m_0(?j,?a,?i)\\
q_1(?g,?g,?g) &\leftarrow m_0(?g,?h,?g)\\
q_1(?f,?f,?f) &\leftarrow m_0(C,?f,?f) \wedge r_0(?f)\\
r_1(?d) &\leftarrow q_0(?d,?e) \wedge s_0(D) \wedge q_1(D,?e,D)\\
s_1(?i) &\leftarrow p_0(?i,?i)\\
s_1(?f) &\leftarrow q_0(?f,?g) \wedge s_1(?h)\\
k_2(?c,?d) &\leftarrow m_1(?e,?d,?c) \wedge k_1(?f) \wedge k_2(?f,?d)\\
k_2(?b,?b) &\leftarrow n_1(E,D,?b) \wedge m_1(?b,E,?b) \wedge q_1(?b,?b,D)\\
k_2(?j,?j) &\leftarrow q_1(?a,?j,?j)\\
k_2(E,E) &\leftarrow l_1(E,E)\\
l_2(?d,?e) &\leftarrow s_1(?d) \wedge n_0(E,?e) \wedge l_2(?e,?e)\\
l_2(C,C) &\leftarrow l_2(C,C) \wedge l_1(E,E)\\
l_2(?j,?j) &\leftarrow p_0(?a,?a) \wedge s_1(?b) \wedge m_0(?c,?b,?j)\\
l_2(?g,?g) &\leftarrow m_0(?h,?g,?i) \wedge m_1(?i,?h,?h) \wedge p_0(?h,?g)\\
m_2(?f) &\leftarrow s_0(?f) \wedge l_1(?g,?h)\\
m_2(B) &\leftarrow k_1(B)\\
n_2(?a) &\leftarrow p_1(?b,?c,?a)\\
n_2(A) &\leftarrow m_1(B,A,E) \wedge k_1(C) \wedge n_1(E,A,E) \wedge q_1(C,A,D)\\
n_2(C) &\leftarrow l_1(E,C) \wedge k_0(B)\\
n_2(E) &\leftarrow r_1(B) \wedge r_0(E) \wedge p_1(B,?i,?j)\\
p_2(?b,A,?b) &\leftarrow q_1(?b,A,?b)\\
p_2(?j,?j,?j) &\leftarrow k_1(?a) \wedge k_0(?a) \wedge l_2(A,?a) \wedge k_2(?j,A)\\
p_2(C,E,E) &\leftarrow l_1(C,B) \wedge q_1(E,E,E)\\
p_2(B,C,A) &\leftarrow r_0(E) \wedge n_1(C,?i,?i) \wedge p_0(B,?i) \wedge k_2(C,A)\\
p_2(E,?g,?h) &\leftarrow r_0(E) \wedge p_1(?g,?h,E)\\
p_2(?c,?d,?d) &\leftarrow p_1(?c,?e,?f) \wedge l_1(?e,?f) \wedge p_2(?c,?d,?c)\\
p_2(E,C,C) &\leftarrow r_1(D) \wedge l_1(E,C)\\
p_2(?j,?j,?j) &\leftarrow m_1(?a,?b,?j) \wedge p_2(?a,?j,?a)\\
p_2(?h,?h,D) &\leftarrow r_1(A) \wedge m_0(?i,?h,D)\\
p_2(?f,?f,?f) &\leftarrow m_1(?g,?g,?f)\\
p_2(D,D,D) &\leftarrow k_1(D) \wedge s_0(D)\\
p_2(?c,?d,?d) &\leftarrow n_1(?e,?d,?e) \wedge p_0(?c,?d) \wedge p_2(?c,?d,?c)\\
p_2(?b,?b,?b) &\leftarrow n_1(D,D,?b)\\
p_2(?a,?a,?a) &\leftarrow q_1(?a,?a,?a)\\
p_2(?h,?i,?i) &\leftarrow k_1(?j) \wedge p_2(E,?h,?i)\\
p_2(?f,E,?g) &\leftarrow n_1(E,?f,A) \wedge q_1(A,?g,?f)\\
p_2(?e,?e,?e) &\leftarrow l_1(?e,D)\\
p_2(?b,?b,?c) &\leftarrow q_1(C,?b,?d) \wedge s_1(C) \wedge s_0(E) \wedge p_2(?b,?d,?b)\\
p_2(?a,?a,?a) &\leftarrow k_1(?a)\\
p_2(?h,?h,?h) &\leftarrow m_1(A,A,?i) \wedge p_2(A,?j,?h)\\
p_2(D,B,B) &\leftarrow p_1(D,B,E)\\
p_2(B,C,C) &\leftarrow p_1(?g,B,B) \wedge n_1(E,E,?g) \wedge q_1(E,C,?g)\\
p_2(?e,?e,?e) &\leftarrow q_1(?f,C,?f) \wedge k_2(?e,C)\\
p_2(?b,?b,?b) &\leftarrow p_0(?b,?b) \wedge r_1(?c) \wedge p_2(?d,?c,?b)\\
p_2(?i,?i,?i) &\leftarrow q_1(?j,?a,?j) \wedge p_2(?j,?j,?a)\\
p_2(A,?h,D) &\leftarrow n_0(?h,D) \wedge m_1(A,?h,D)\\
p_2(?g,?g,?g) &\leftarrow s_1(?g) \wedge k_1(?g)
\end{aligned}
$$

$$
\begin{aligned}
p_2(A,C,B) &\leftarrow l_1(E,C) \wedge l_2(E,B) \wedge r_1(E) \wedge m_1(D,A,C)\\
p_2(?d,?d,?d) &\leftarrow q_1(?e,?e,?e) \wedge p_1(?d,?f,?d)\\
p_2(?c,E,?c) &\leftarrow n_1(?c,E,E)\\
p_2(?a,?a,?a) &\leftarrow n_1(?b,?b,?b) \wedge p_0(?a,?a)\\
p_2(A,A,A) &\leftarrow p_1(E,E,A)\\
p_2(?i,?i,?i) &\leftarrow r_1(?j) \wedge r_0(?i)\\
p_2(?h,?h,?h) &\leftarrow n_2(?h) \wedge k_1(E)\\
p_2(?f,?f,?f) &\leftarrow l_1(?g,?f)\\
p_2(?d,?e,?d) &\leftarrow m_1(?e,?d,?e)\\
q_2(?e,?f,?f) &\leftarrow k_0(?f) \wedge p_1(?e,?e,?e)\\
q_2(?b,?b,?b) &\leftarrow q_0(?c,?b) \wedge n_1(?c,?b,?d)\\
q_2(?j,?j,?j) &\leftarrow k_1(?a) \wedge n_1(?j,?j,?a)\\
q_2(D,A,A) &\leftarrow q_2(D,C,A) \wedge s_1(C) \wedge q_0(E,C)\\
q_2(?i,?i,?i) &\leftarrow p_1(?i,?i,?i)\\
q_2(?f,?g,?f) &\leftarrow p_1(?f,?f,?h) \wedge n_1(?g,?f,?h) \wedge q_2(?g,?h,?f)\\
q_2(?d,C,?e) &\leftarrow k_1(?e) \wedge l_0(C) \wedge l_2(?e,?d)\\
q_2(?b,?b,?b) &\leftarrow q_1(?c,C,?b)\\
q_2(?i,?i,?i) &\leftarrow n_1(?j,D,?a) \wedge k_1(?i) \wedge q_2(?a,?a,?j)\\
q_2(?g,?g,?h) &\leftarrow l_1(?h,?g)\\
q_2(?c,?d,?c) &\leftarrow r_1(?d) \wedge m_0(?e,?f,?c) \wedge k_0(?d) \wedge q_2(?d,?d,?d)\\
r_2(?g) &\leftarrow r_1(?g) \wedge l_0(?g)\\
s_2(?h) &\leftarrow q_2(B,?h,B) \wedge s_1(B)\\
s_2(D) &\leftarrow s_1(A) \wedge s_0(D)\\
s_2(D) &\leftarrow r_1(D) \wedge s_1(D)\\
k_3(?j,?a,?j) &\leftarrow s_1(?a) \wedge p_2(?b,?a,?c) \wedge n_0(?j,?c)\\
k_3(?h,?h,?h) &\leftarrow s_1(?h) \wedge q_2(D,?i,D) \wedge s_2(?i)\\
k_3(?f,?f,?g) &\leftarrow k_2(?g,?f)\\
k_3(C,C,C) &\leftarrow s_2(E) \wedge k_2(C,E)\\
k_3(?c,?c,?c) &\leftarrow p_2(?d,?e,?d) \wedge m_1(?c,?c,?e)\\
k_3(?a,?a,?a) &\leftarrow l_2(?b,B) \wedge k_1(?a)\\
k_3(C,C,C) &\leftarrow k_0(A) \wedge r_2(C)\\
k_3(?i,?j,?j) &\leftarrow l_1(?j,?i) \wedge k_3(?i,?j,?j)\\
k_3(?f,?f,?f) &\leftarrow p_2(?g,?h,E) \wedge s_1(?g) \wedge k_3(?f,?g,?g)\\
k_3(?b,?b,?c) &\leftarrow p_1(?c,?d,?b) \wedge m_2(?e) \wedge m_2(?d)\\
k_3(?g,?g,?h) &\leftarrow q_0(?i,?h) \wedge k_2(?g,?j) \wedge k_3(?h,?a,?j)\\
k_3(D,D,D) &\leftarrow p_1(A,D,B) \wedge r_2(A) \wedge l_2(E,B)\\
k_3(A,A,A) &\leftarrow r_2(A)\\
k_3(?e,?e,?e) &\leftarrow p_2(?f,?e,?e)\\
k_3(?c,?d,?c) &\leftarrow p_2(?d,?c,?c)\\
k_3(?j,?j,?j) &\leftarrow p_0(?a,?j) \wedge k_3(?j,?j,?j) \wedge k_3(?a,?j,?b)\\
k_3(?i,?i,?i) &\leftarrow r_2(C) \wedge l_1(B,?i)\\
k_3(?e,?e,?e) &\leftarrow m_2(?f) \wedge l_1(?g,?h) \wedge s_2(?e) \wedge k_3(?g,?h,?g)\\
k_3(?d,?d,?d) &\leftarrow n_2(?d)\\
k_3(?c,?c,?c) &\leftarrow l_0(?c) \wedge r_2(E) \wedge r_0(E)\\
k_3(?b,?b,?b) &\leftarrow m_2(?b)\\
k_3(?i,?i,?i) &\leftarrow r_1(?i) \wedge p_2(?j,?a,?a)\\
k_3(C,C,C) &\leftarrow r_2(C)
\end{aligned}
$$

$$
\begin{aligned}
l_3(?g, ?h) &\leftarrow r_0(?g) \wedge p_2(?g, ?h, ?g)\\
l_3(?d, ?d) &\leftarrow p_1(?d, ?d, ?e) \wedge p_2(?e, ?f, ?d)\\
l_3(?c, ?c) &\leftarrow n_2(?c) \wedge m_2(B)\\
l_3(?b, ?b) &\leftarrow r_2(?b)\\
l_3(?i, ?i) &\leftarrow n_2(?j) \wedge l_1(?a, ?i) \wedge l_3(?a, ?a)\\
l_3(?g, ?g) &\leftarrow s_2(?h) \wedge l_1(?g, ?g)\\
l_3(D, D) &\leftarrow k_2(A, D)\\
l_3(?d, ?d) &\leftarrow k_3(?e, ?d, ?d) \wedge l_2(?f, ?f)\\
l_3(C, C) &\leftarrow k_2(B, C)\\
l_3(D, C) &\leftarrow s_2(D) \wedge k_3(A, C, A) \wedge r_0(B)\\
m_3(?j, ?a, ?j) &\leftarrow m_0(?b, ?b, ?a) \wedge l_2(?c, ?j) \wedge m_0(?j, ?c, ?c) \wedge s_2(?b)\\
m_3(?g, ?g, ?g) &\leftarrow k_2(?h, ?i) \wedge m_3(?g, ?i, ?g) \wedge n_0(?i, A) \wedge l_2(A, A)\\
m_3(?c, ?c, ?c) &\leftarrow q_0(?d, ?e) \wedge s_0(?f) \wedge s_2(?e) \wedge r_2(?c)\\
m_3(?j, ?a, ?a) &\leftarrow n_2(?j) \wedge m_2(?a) \wedge m_3(?b, ?j, ?b)\\
m_3(B, B, B) &\leftarrow q_2(A, B, A)\\
m_3(C, B, D) &\leftarrow l_1(D, B) \wedge m_2(D) \wedge q_2(B, C, D)\\
m_3(?h, ?i, ?h) &\leftarrow r_2(?h) \wedge k_2(C, ?i)\\
m_3(?g, ?g, ?g) &\leftarrow l_2(?g, ?g) \wedge n_2(?g)\\
m_3(?e, ?e, ?e) &\leftarrow n_2(?e) \wedge m_2(?f)\\
m_3(?d, E, E) &\leftarrow n_2(E) \wedge p_2(?d, E, E)\\
m_3(?b, ?b, ?c) &\leftarrow r_2(?c) \wedge k_3(?b, ?c, ?b)\\
m_3(?a, ?a, ?a) &\leftarrow n_2(?a)\\
m_3(?j, C, ?j) &\leftarrow s_2(C) \wedge q_2(?j, C, C)\\
m_3(?i, ?i, ?i) &\leftarrow p_2(?i, ?i, ?i)\\
m_3(B, B, B) &\leftarrow l_2(A, B)\\
n_3(?d) &\leftarrow p_2(?e, ?f, ?d)\\
p_3(?c, ?d, ?e) &\leftarrow q_2(?f, D, ?c) \wedge k_2(?d, ?e)\\
p_3(?j, ?a, ?b) &\leftarrow r_2(?a) \wedge k_3(?a, ?b, ?j)\\
p_3(?i, D, E) &\leftarrow l_3(B, E) \wedge p_2(D, B, C) \wedge n_3(?i) \wedge q_2(?i, D, ?i)\\
p_3(?h, ?h, ?h) &\leftarrow n_2(?h)\\
p_3(?e, ?e, ?e) &\leftarrow l_1(?f, ?f) \wedge l_3(?f, ?e) \wedge p_3(?g, ?g, ?f)\\
p_3(?d, ?d, ?d) &\leftarrow l_2(?d, ?d)\\
p_3(?a, ?a, ?a) &\leftarrow n_2(?a) \wedge q_2(?b, ?c, ?a) \wedge s_1(?b)\\
p_3(?i, ?i, ?i) &\leftarrow p_2(?j, ?i, ?i) \wedge n_3(?i)\\
p_3(?h, ?h, ?h) &\leftarrow k_1(?h) \wedge n_2(?h)\\
p_3(?e, ?e, ?e) &\leftarrow k_1(?e) \wedge q_2(?f, ?g, ?e)\\
p_3(?a, ?b, ?b) &\leftarrow m_3(?b, ?c, ?d) \wedge p_2(?a, ?b, ?d)\\
p_3(?h, ?h, ?h) &\leftarrow q_0(?i, ?h) \wedge k_2(?j, ?j)\\
p_3(B, C, B) &\leftarrow k_2(C, B)\\
p_3(E, B, E) &\leftarrow m_3(E, ?g, E) \wedge q_2(?g, ?g, B)\\
q_3(?g, ?h) &\leftarrow q_2(?i, ?g, ?h) \wedge n_0(?i, ?g)\\
q_3(?e, ?e) &\leftarrow p_2(?f, ?e, ?e) \wedge q_3(?f, ?e)\\
q_3(?b, ?c) &\leftarrow n_1(?d, ?b, ?c) \wedge s_2(?b) \wedge q_3(?c, ?b)\\
q_3(?i, ?i) &\leftarrow r_2(?i) \wedge s_1(?j) \wedge l_2(?a, ?a)\\
q_3(?g, ?g) &\leftarrow m_0(?h, D, ?h) \wedge k_1(?g) \wedge r_2(D) \wedge q_3(?h, ?g)\\
r_3(?g, ?h, ?h) &\leftarrow s_2(?h) \wedge l_2(C, ?g)\\
r_3(?d, ?d, ?d) &\leftarrow l_1(?e, ?f) \wedge n_1(?f, ?f, ?f) \wedge r_2(?d)
\end{aligned}
$$

$$
\begin{aligned}
r_3(?a,?a,?a) &\leftarrow p_1(?b,?c,?a) \wedge l_2(?c,?b) \wedge r_3(?a,?b,?a)\\
r_3(?i,?i,?i) &\leftarrow m_0(D,?j,?i) \wedge r_3(?i,?i,?j)\\
r_3(?h,?h,?h) &\leftarrow s_2(?h)\\
r_3(?f,?f,?f) &\leftarrow l_2(?g,?f)\\
r_3(?e,?e,?e) &\leftarrow r_2(?e)\\
r_3(?b,?c,?b) &\leftarrow p_2(?b,?d,?c)\\
r_3(?h,?h,?i) &\leftarrow m_2(?i) \wedge m_3(?j,B,?h) \wedge r_3(?i,?a,?a)\\
r_3(A,A,E) &\leftarrow k_2(A,A) \wedge q_2(?g,E,?g) \wedge m_2(B) \wedge m_3(A,?g,?g)\\
r_3(?f,B,?f) &\leftarrow r_0(?f) \wedge p_2(B,?f,B) \wedge l_2(?f,?f)\\
r_3(?c,?c,?c) &\leftarrow p_3(?d,?c,?e) \wedge r_3(?d,?d,?d)\\
r_3(?j,?a,?b) &\leftarrow k_2(?a,?b) \wedge r_2(?b) \wedge r_3(?b,?j,?j)\\
s_3(?i,?j) &\leftarrow q_2(?a,?i,?a) \wedge s_2(?i) \wedge m_0(?a,?b,?j)\\
k_4(C) &\leftarrow n_0(C,D) \wedge q_3(E,B) \wedge n_3(E)\\
k_4(?e) &\leftarrow k_3(?f,?f,?f) \wedge n_0(?g,?f) \wedge k_4(?e)\\
k_4(E) &\leftarrow q_3(?c,?c) \wedge q_1(A,A,?d) \wedge r_3(?c,E,?d)\\
l_4(?j) &\leftarrow p_3(?a,?b,?j)\\
l_4(?h) &\leftarrow m_0(?i,?h,?h) \wedge l_4(?i)\\
m_4(?e,?f) &\leftarrow l_2(?g,?f) \wedge s_3(A,?e)\\
m_4(?c,?c) &\leftarrow p_3(?d,?d,?d) \wedge m_3(?c,?c,?d) \wedge m_4(?c,?c)\\
n_4(?j,?a) &\leftarrow p_3(?j,?a,?a) \wedge n_4(?j,?j)\\
n_4(D,D) &\leftarrow k_3(C,C,E) \wedge q_1(D,D,D)\\
n_4(E,E) &\leftarrow l_3(B,A) \wedge p_3(B,E,A)\\
n_4(?h,?h) &\leftarrow k_4(?i) \wedge m_3(?h,?h,?h)\\
p_4(?g,?g,?h) &\leftarrow r_0(?g) \wedge r_3(?h,?g,?h)\\
p_4(?d,?d,?d) &\leftarrow q_3(?e,?f) \wedge n_3(?d)\\
p_4(?b,?c,?b) &\leftarrow k_3(?b,?b,?c)\\
p_4(?h,?i,?i) &\leftarrow r_3(?i,?i,?h) \wedge p_4(?j,?j,?a)\\
p_4(?d,?d,?d) &\leftarrow l_4(?d) \wedge n_0(?d,?e) \wedge p_4(?f,?g,?f)\\
p_4(?a,?a,?a) &\leftarrow m_3(?b,?c,?a) \wedge p_4(?a,?c,?a)\\
p_4(?i,?i,?i) &\leftarrow p_3(?j,?j,?i)\\
p_4(?f,?f,?f) &\leftarrow k_3(?g,?h,?h) \wedge n_4(?f,?h) \wedge p_1(?h,?g,?f)\\
p_4(?c,?c,?c) &\leftarrow q_3(?d,?e) \wedge n_4(?e,?c) \wedge l_3(?d,?d)\\
p_4(C,C,?b) &\leftarrow n_3(A) \wedge m_3(?b,C,A)\\
q_4(?b,?c) &\leftarrow k_3(?d,?d,?b) \wedge q_2(?e,?c,?b) \wedge m_3(?e,?f,?e)\\
q_4(?i,?i) &\leftarrow r_1(?i) \wedge l_4(?j) \wedge q_4(?j,?a)\\
q_4(B,B) &\leftarrow k_3(C,E,B) \wedge l_1(B,C)\\
r_4(?g) &\leftarrow n_3(?g) \wedge q_3(?h,?i) \wedge p_0(?j,?g)\\
s_4(?a) &\leftarrow p_3(?b,?c,?d) \wedge l_1(?a,?c)\\
k_5(?e) &\leftarrow s_4(?f) \wedge r_3(?g,?e,?e)\\
k_5(B) &\leftarrow s_4(E) \wedge n_1(B,B,B)\\
l_5(?h) &\leftarrow q_4(?i,?i) \wedge k_1(?h)\\
m_5(?d,?e) &\leftarrow r_0(?d) \wedge p_4(?d,?f,?e)\\
m_5(?c,?c) &\leftarrow k_4(?c)\\
m_5(?b,?b) &\leftarrow s_4(?b) \wedge m_4(E,E)\\
m_5(?j,?j) &\leftarrow s_4(?a) \wedge r_0(?j)\\
n_5(?b,?c) &\leftarrow q_4(?d,?e) \wedge n_4(?b,?e) \wedge p_4(?f,?f,?c) \wedge n_5(?f,?c)\\
n_5(?j,?j) &\leftarrow m_0(?a,?j,?a) \wedge n_5(?a,?j)
\end{aligned}
$$

$$
\begin{aligned}
n_5(?h, ?h) &\leftarrow n_4(?i, ?h)\\
n_5(?e, ?e) &\leftarrow p_0(?e, ?f) \wedge m_4(?g, ?g) \wedge k_5(D)\\
n_5(D, D) &\leftarrow p_4(C, D, C)\\
n_5(B, E) &\leftarrow r_3(D, B, C) \wedge n_4(B, E)\\
n_5(?c, ?c) &\leftarrow q_4(?d, ?c)\\
n_5(?b, ?b) &\leftarrow r_4(?b)\\
n_5(?a, ?a) &\leftarrow s_1(?a) \wedge k_4(?a)\\
n_5(?h, ?h) &\leftarrow p_4(?i, ?h, ?h) \wedge s_1(?h) \wedge p_4(B, B, ?j)\\
n_5(D, ?g) &\leftarrow k_3(D, ?g, A) \wedge n_4(?g, D)\\
p_5(?e, ?e, ?f) &\leftarrow s_4(?f) \wedge l_2(?e, ?e)\\
p_5(?b, ?c, ?c) &\leftarrow l_2(?b, ?d) \wedge p_5(?b, ?d, ?b) \wedge m_4(?d, ?c)\\
p_5(?i, ?j, ?i) &\leftarrow q_4(?j, ?a) \wedge s_1(?i)\\
p_5(B, B, B) &\leftarrow p_4(?g, ?g, ?h) \wedge k_5(B)\\
q_5(?j, ?a) &\leftarrow s_4(?j) \wedge m_2(?a)\\
q_5(A, A) &\leftarrow r_4(A)\\
q_5(?i, ?i) &\leftarrow l_4(?i)\\
q_5(?h, ?h) &\leftarrow q_4(?h, ?h)\\
q_5(?g, ?g) &\leftarrow m_4(?g, ?g)\\
r_5(?c, ?d) &\leftarrow s_4(?c) \wedge k_0(B) \wedge n_3(?d)\\
r_5(?b, ?b) &\leftarrow k_4(?b)\\
s_5(?h) &\leftarrow l_4(?h) \wedge r_4(?i)\\
s_5(?e) &\leftarrow k_4(?e) \wedge s_3(?e, ?f) \wedge l_5(?g) \wedge s_5(?g)
\end{aligned}
$$

Problem Set 1, Quasi-Uniform Distribution

$n_1(E, E, ?x)$	$m_4(B, ?x)$	$r_1(E)$	$n_3(E)$	$l_2(B, E)$
$m_5(D, ?x)$	$r_1(A)$	$q_2(?x, D, ?y)$	$q_3(B, D)$	$l_3(E, C)$
$l_0(A)$	$k_2(A, ?x)$	$k_3(?x, ?x, B)$	$s_3(B, B)$	$k_2(C, B)$
$s_3(C, C)$	$m_3(C, E, C)$	$s_4(B)$	$n_3(A)$	$p_1(?x, A, ?x)$
$k_2(?x, D)$	$n_2(D)$	$s_3(A, D)$	$n_2(D)$	$s_3(?x, B)$
$l_5(E)$	$q_0(C, D)$	$m_4(A, A)$	$r_3(C, B, C)$	$n_2(B)$
$n_4(A, E)$	$s_5(D)$	$l_2(D, D)$	$r_2(C)$	$r_0(B)$
$q_1(E, C, E)$	$m_0(?x, ?y, E)$	$k_2(C, A)$	$m_0(?x, ?y, B)$	$m_5(D, D)$
$s_4(D)$	$q_5(?x, C)$	$n_5(B, B)$	$s_5(D)$	$k_5(E)$
$q_4(A, E)$	$k_1(B)$	$q_3(?x, B)$	$p_1(D, ?x, D)$	$k_1(E)$
$m_3(?x, A, A)$	$l_0(C)$	$n_0(D, ?x)$	$k_5(B)$	$k_0(E)$
$s_1(E)$	$k_0(B)$	$r_2(C)$	$r_0(E)$	$s_2(B)$
$l_3(A, A)$	$s_4(C)$	$n_1(C, D, D)$	$l_4(B)$	$r_5(D, D)$
$p_4(D, A, A)$	$n_5(A, E)$	$m_5(C, ?x)$	$m_0(?x, E, A)$	$s_4(E)$
$k_0(E)$	$s_4(E)$	$s_2(E)$	$p_1(A, B, B)$	$s_4(E)$
$n_2(E)$	$s_5(C)$	$s_4(D)$	$r_1(D)$	$l_5(C)$
$s_3(C, C)$	$k_2(E, D)$	$q_1(E, A, E)$	$q_5(E, E)$	$r_4(B)$
$r_5(C, C)$	$r_1(E)$	$s_1(D)$	$s_3(D, B)$	$n_1(?x, C, ?y)$
$p_0(B, C)$	$s_3(A, C)$	$s_5(B)$	$k_0(E)$	$m_5(E, A)$
$n_3(A)$	$m_3(?x, ?y, A)$	$l_1(E, E)$	$r_4(A)$	$k_2(E, ?x)$
$k_5(E)$	$n_5(A, A)$	$s_1(B)$	$l_2(A, B)$	$r_1(E)$

$n_1(?x, D, ?x)$	$s_5(B)$	$s_1(A)$	$l_5(E)$	$k_2(B, C)$
$p_3(?x, D, ?y)$	$n_5(C, C)$	$l_4(E)$	$n_4(?x, A)$	$l_5(E)$
$q_4(D, B)$	$p_4(?x, C, ?y)$	$r_3(A, ?x, D)$	$p_3(?x, ?y, B)$	$k_5(E)$
$m_2(D)$	$n_5(E, A)$	$s_2(B)$	$s_1(C)$	$l_4(B)$
$q_3(D, ?x)$	$k_4(C)$	$m_0(D, ?x, D)$	$m_3(B, C, B)$	$n_5(C, ?x)$
$k_5(E)$	$q_2(E, C, ?x)$	$s_5(C)$	$n_5(?x, D)$	$r_4(A)$

Problem Set 2, Skewed Distribution

$l_3(B, A)$	$l_2(A, A)$	$m_1(A, A, ?x)$	$l_3(B, E)$	$k_2(E, D)$
$l_3(B, C)$	$l_3(?x, E)$	$m_1(C, ?x, ?x)$	$n_2(D)$	$l_3(B, C)$
$l_3(B, B)$	$p_4(E, C, E)$	$q_1(?x, A, C)$	$l_3(E, A)$	$l_3(E, B)$
$m_4(?x, D)$	$q_1(B, B, B)$	$l_3(C, C)$	$n_2(B)$	$l_2(E, B)$
$l_3(?x, D)$	$l_3(C, ?x)$	$q_1(A, E, ?x)$	$q_1(E, A, E)$	$n_2(E)$
$l_2(C, ?x)$	$l_2(B, B)$	$l_3(A, A)$	$m_1(D, ?x, ?y)$	$q_1(E, E, E)$
$q_0(D, ?x)$	$q_1(D, ?x, ?y)$	$m_4(E, ?x)$	$m_1(A, A, A)$	$l_3(?x, B)$
$p_1(E, C, E)$	$l_3(D, D)$	$n_3(A)$	$n_2(D)$	$l_3(A, ?x)$
$l_3(B, E)$	$l_3(?x, C)$	$n_2(A)$	$p_3(?x, ?x, A)$	$q_1(?x, D, ?y)$
$q_1(D, D, C)$	$l_3(E, E)$	$l_3(?x, C)$	$m_1(?x, ?x, E)$	$q_1(A, B, A)$
$s_3(B, D)$	$q_1(?x, B, ?y)$	$l_3(?x, D)$	$l_3(?x, D)$	$q_1(?x, ?x, D)$
$n_2(C)$	$p_1(?x, ?y, D)$	$l_3(C, ?x)$	$l_3(A, A)$	$l_3(D, ?x)$
$p_4(?x, C, ?y)$	$s_3(B, A)$	$m_1(E, A, A)$	$n_2(C)$	$q_1(C, ?x, ?y)$
$m_1(C, ?x, C)$	$q_1(D, D, C)$	$m_1(?x, C, ?x)$	$l_3(E, E)$	$l_3(E, C)$
$l_3(E, E)$	$m_1(C, B, C)$	$p_4(D, E, ?x)$	$q_1(A, B, A)$	$p_1(B, C, B)$
$l_3(A, ?x)$	$q_1(D, D, D)$	$m_1(E, B, B)$	$n_2(E)$	$p_1(?x, D, ?x)$
$l_2(A, A)$	$m_1(A, D, D)$	$q_1(E, B, E)$	$m_4(E, B)$	$q_1(B, B, B)$
$q_0(D, B)$	$q_1(A, C, A)$	$p_1(D, D, A)$	$p_0(C, B)$	$m_1(?x, E, E)$
$n_2(B)$	$q_1(?x, B, ?y)$	$q_1(A, D, D)$	$l_3(C, ?x)$	$s_4(A)$
$p_1(B, ?x, B)$	$s_3(B, ?x)$	$m_1(?x, B, ?y)$	$n_2(E)$	$l_3(E, E)$
$q_1(?x, C, C)$	$s_3(B, A)$	$m_1(D, C, C)$	$q_1(?x, B, ?y)$	$l_3(E, B)$
$q_1(D, D, A)$	$n_2(A)$	$l_3(?x, C)$	$p_3(B, E, E)$	$l_3(E, B)$
$p_4(?x, ?y, C)$	$l_0(C)$	$l_3(B, E)$	$l_3(E, ?x)$	$m_1(D, E, E)$
$l_3(E, ?x)$	$p_4(A, ?x, ?y)$	$p_4(E, C, E)$	$n_2(B)$	$p_1(?x, D, D)$
$p_1(?x, ?y, D)$	$n_2(B)$	$l_3(B, C)$	$l_3(?x, E)$	$p_4(D, ?x, A)$
$q_1(?x, ?y, E)$	$p_4(B, A, A)$	$p_4(B, B, C)$	$q_0(?x, D)$	$q_1(B, ?x, ?y)$
$l_3(B, E)$	$l_3(A, A)$	$q_1(B, D, B)$	$q_1(D, B, B)$	$l_3(E, A)$