The Utility of Knowledge in Inductive Learning

Michael Pazzani
pazzani@ics.uci.edu
Dennis Kibler
kibler@ics.uci.edu

Technical Report 90–18

June 25, 1990
Revised November 12, 1990

Submitted to *Machine Learning*.

This research is partially supported by NSF Grant IRI-8908260. We would like to thank Ross Quinlan for his advice on FOIL, Dan Hirschberg for deriving the *new* recurrence, and Cliff Brunk, Tim Cain, Caroline Ehrlich, Ross Quinlan, Wendy Sarrett and Glenn Silverstein for reviewing a draft of this paper.

The Utility of Knowledge in Inductive Learning

Michael Pazzani
pazzani@ics.uci.edu

Dennis Kibler
kibler@ics.uci.edu

Department of Information & Computer Science
University of California, Irvine
Irvine, CA 92717 U.S.A.
(714) 856-5951

Running Head: Knowledge in Inductive Learning

Abstract

In this paper, we demonstrate how different forms of background knowledge can be integrated with an inductive method for generating constant-free Horn clause rules. Furthermore, we evaluate, both theoretically and empirically, the effect that these types of knowledge have on the cost of learning a rule and on the accuracy of a learned rule. Moreover, we demonstrate that a hybrid explanation-based and inductive learning method can advantageously use an approximate domain theory, even when this theory is incorrect and incomplete.

1 Introduction

Most existing systems that combine empirical and explanation-based learning severely restrict the complexity of the language for expressing the concept definition. For example, some systems require that the concept definition be expressed in terms of attribute-value pairs (Lebowitz, 1986; Danyluk, 1989). Others effectively restrict the concept definition language to that of propositional calculus, by only allowing unary predicates (Hirsh, 1989; Mooney & Ourston, 1989; Katz, 1989; Shavlik & Towell, 1989; Pazzani, 1989; Sarrett & Pazzani, 1989). The few systems that allow relational concept definitions (e.g., OCCAM (Pazzani, 1990), IOE (Flann & Dietterich, 1989), ML-SMART (Bergadano, Giordana, & Ponsero, 1989)) place strong restrictions on the form of induction and the initial knowledge that is provided to the system. The restricted concept definitions languages that are usually required by the empirical learning component, reduce the applicability of the integrated learning system.

A recent advance in concept formation, FOIL (Quinlan, 1989; Quinlan, 1990) learns constant-free Horn clauses, a useful subset of first-order predicate calculus. In this paper, we analyze the complexity of FOIL in terms of the size of hypothesis space generated and tested during learning. We describe how FOIL can be extended to use a variety of types of background knowledge to increase the class of problems that can be solved, to decrease the hypothesis space explored, and to increase the accuracy of learned rules.

We introduce a new learning system called FOCL (First Order Combined Learner) that uses FOIL's information-based metric to evaluate extensions to a (possibly null) hypothesis of a concept definition. The extensions may be proposed either by an inductive component or by an explanation-based component. We demonstrate that FOCL can utilize knowledge of the predicates, approximate concept definitions and background predicates to learn Horn clause concept definitions more accurately, more robustly, and more efficiently than without such background knowledge.

Given a set of a examples and a correct domain theory, the output of FOCL is similar to that produced by m-EBG (Flann & Dietterich, 1989). Without any background knowledge, FOCL operates like FOIL. The interesting issues arise when FOCL is given an incomplete or incorrect domain theory. In this case, some clauses of a rule may be learned purely analytically, others may be learned purely empirically, and some clauses may be learned by a combination of methods (i.e., some literals of a single clause are added empirically and others are added analytically).

A secondary goal of this paper is to create a taxonomy of various types of background knowledge and show the effect that each type of knowledge has on the size of the hypothesis space or the portion of the hypothesis space searched. We take a broad view of prior knowledge that includes typing information, both extensionally and intensionally defined predicates, and an initial, approximate definition of the concept to be learned.

This taxonomy can be used to illuminate the similarities and differences between the types of background knowledge used in a variety of systems including COBWEB (Fisher, 1987), CIGOL (Muggleton & Buntine, 1988), and OCCAM (Pazzani, 1990). We demonstrate the effects of each form of knowledge with a complexity analysis of FOIL and with a series of experiments on learning concepts from the domains of list relations and chess end game relations.

We selected list relations primarily to illustrate the mechanism. Chess end games were selected because this was the most difficult problem on which FOIL has been tested. In addition, a domain theory for this particular chess problem is relatively succinct and amenable to systematic experimentation by mutating the correct domain theory. In fact, FOCL has been tested on a variety of problems, including a number of standard EBL problems, and a larger problem that includes a domain theory describing when a student loan is required to be repaid (Pazzani & Brunk, 1990).

2 Background: FOIL

In this section, we present a brief discussion of FOIL that will allow us to analyze its complexity and that of FOCL. We begin by introducing some definitions. A *predicate* is a Boolean function. Predicates can be defined *extensionally*, as a list of tuples for which the predicate is true, or *intensionally*, as a set of (Horn) clauses for computing whether the predicate is true. A *literal* is a predicate or its negation, i.e., the Boolean function that is the Boolean complement of the predicate. Literals that are unnegated predicates are called *positive literals*. Literals that are negation of predicates are called *negative literals*. A *clause body* is a conjunction of literals. A *Horn clause* consists of a clause head, which is a predicate, and a clause body. It has the form $P \leftarrow L_1, L_2, ...$ where each L_i is a literal. A *rule* for P is a collection of Horn clauses each with the head P.

For completeness, we will define the semantics of a rule. In general, a *k-tuple* is a finite sequence of k constants, denoted by $<a_1, \ldots, a_k>$. The meaning of rule for a k-arity predicate is the set of k-tuples that satisfy the predicate. A tuple *satisfies* a rule if it satisfies one of the Horn clauses that define the rule. A tuple *satisfies* a Horn clause if there is a mapping ϕ of the variables of the head onto the tuple and an extension ϕ' of all the variables in the positive literals of the clause body into constants such that for each literal in the clause body, the bindings resulting from ϕ' result in a satisfiable literal. Note that a negative literal is *satisfiable* if there does not exist any bindings for the remaining variables (if any) that make the predicate satisfiable.

Given positive and negative examples of some concept, and a set of extensionally defined background predicates, FOIL inductively generates a logical concept definition or rule for the concept. FOIL and FOCL share the restriction that the induced rule must not involve any constants or function symbols,[1] but does allow negated

[1]For some problems, this is not a severe restriction. For example, $color(X, red)$ may be repre-

Table 1: FOIL Design I

```
Let POS be the positive examples.
Let NEG be the negative examples.
Set NewClauseBody to empty.
Until POS is empty do:
  Separate: (begins new clauses)
    Remove from POS all examples that satisfy the NewClauseBody.
    Reset NEG to the original negative examples.
    Reset NewClauseBody to empty.
  Until NEG is empty do:
      Conquer: (build clause body)
        Choose a literal L.
        Conjoin L to NewClauseBody.
        Remove from NEG examples that do not satisfy L.
```

predicates. In a restricted way, FOIL also allows the use of the predicate to be learned. In this way, FOIL can learn some recursive concepts. Like ID3 (Quinlan, 1986), FOIL is a non-incremental learner that hill climbs using a metric based on information theory to construct a rule that covers the data. Pagallo and Haussler (1990) introduced the idea of separate-and-conquer to define their GROVE and GREEDY3 algorithms. Unlike ID3 and like AQ (Michalski, 1980), FOIL uses this separate-and-conquer approach rather than a divide-and-conquer approach.

The separate stage of the algorithm begins a new clause while the conquer stage constructs a conjunction of literals to serve as the body of the clause. Each clause describes some subset of the positive examples and no negative examples. Note that, in effect, FOIL has two operators: start a new, empty clause, and add a literal to the end of the current clause. FOIL adds literals to the end of the current clause until no negative example is covered by the clause, and starts new clauses until all positive examples are covered by some clause.

In order to define FOIL's algorithm, we need to be a bit more careful about what an example is. For example, suppose FOIL's task is to learn the relation $grandfather(X, Y)$ given the relations $father(X, Y)$ and $parent(X, Y)$, defined extensionally. Furthermore, suppose that the current clause (NewClauseBody in Table 1) is $grandfather(X, Y) \leftarrow parent(X, Z)$. Extensions of this clause can be achieved by conjoining the body with any of the literals *father(X,X), father(Y,Z), father(U,Y), parent(Y,Z), parent(Y,Y),* as well as many others. From this example, we see that to create a literal to extend a clause, not only must a predicate-name

sented as $color(X, Y), red(Y)$, although this representation can greatly increase the search cost.

be selected, but also a particular set of variables for the predicate-name. We call the choice of variables for a predicate-name a *variablization* of the predicate. If the variable chosen already occurs in an unnegated literal of the clause (i.e., in either the head or the current body), then the variable is called *old*. Otherwise, the variable is called *new*. One restriction that FOIL and FOCL place on literals is that they contain at least one old variable.

If an extension of a clause is formed by conjoining a literal that uses only old variables, then the new set of positive and negative examples is the subset of old positive and negative examples that satisfy the additional predicate. As expected, these examples retain their same classifications as positive or negative. The situation is much different if the extension of the clause involves new variables.

Suppose FOIL extends a clause $grandfather(X, Y) \leftarrow true$ by conjoining the literal $parent(X, Z)$, introducing the new variable Z. Now the positive examples consist of those of values $<X, Y, Z>$ such that $grandfather(X, Y)$ is true and $parent(X, Z)$ is true. To reinforce the notion that these examples are very different from the original positive examples, and following the language of Quinlan, we will call these *positive tuples*. For a given pair $<X, Y>$ there may be zero or more values of Z such that $parent(X, Z)$ is *true*. Similarly, the set of *negative tuples* consists of those values of $<X, Y, Z>$ such that $grandfather(X, Y)$ is false, but $parent(X, Z)$ is true. In effect, an example is an ordered *tuple* of bindings for the variables of the clause. When a new variable is introduced, the tuples are extended to include values for that variable.

With this understanding, we can elaborate the original algorithm in Table 2. For simplicity, we refer to the original positive examples as positive tuples.

At a high level of abstraction, FOIL is quite simple. It uses hill climbing to add the literal with the maximum information gain to a clause. For each variablization of each predicate P, FOIL measures the information gain. In order to select the literal with maximum information gain, it is necessary to know how many of the current positive and negative tuples are satisfied by the variablizations of every extensionally defined predicate.[2]

3 Analysis of FOIL

In general, the cost to do a hill-climbing search, such as FOIL and FOCL carry out, is the branching factor times the depth at which a solution is found. Usually the branching factor, while not constant, is at least reasonably bounded. In FOIL,

[2]The information gain metric used by FOIL is

$$Gain(Literal) = T^{++} * (log_2(P_1/P_1 + N_1) - log_2(P_0/P_0 + N_0))$$

where P_0 and N_0 are the number of positive and negative tuples before adding the literal to the clause, P_1 and N_1 are the number of positive and negative tuples after adding the literal to the clause, and T^{++} is the number of positive tuples before adding the literal that have at least one corresponding extension in the positive tuples after adding the literal (Quinlan, 1990).

Table 2: FOIL Design II

```
Let POS be the positive tuples.
Let NEG be the negative tuples.
Set NewClauseBody to empty.
Until POS is empty do:
  Separate: (begins new clauses)
    Remove from POS all tuples that satisfy the NewClauseBody.
    Reset Old to be those variables used in P.
    Reset NEG to the original negative examples.
    Reset NewClauseBody to empty.
    Until NEG is empty do:
    Conquer: (refines clause body)
      Choose a predicate P.
      Choose a variablization of the predicate.
      Compute information gain of P and its negation.
      Choose literal L with most information gain.
      Conjoin the literal with NewClauseBody.
      Add any new variables to Old.
      Let POS be all extensions of POS that are satisfied by the literal.
      Let NEG be all extensions of NEG that are satisfied by the literal.
```

the branching factor grows dramatically, roughly exponentially in the arity of the available predicates, the arity of the predicate to be learned, and the length of the clause that is being learned. In this section, we make these statements precise.

To begin, we estimate the cost of adding a single literal to a clause. There are two reasonable measures we might use to estimate this cost. One measure, we call the *theory-cost*, indicates the number of different literals that can be chosen to extend the body of the given clause. The second measure, called the *evaluation-cost*, measures the cost of computing the information gain of each literal. Note, the evaluation-cost is a function of the number of training examples, while the theory-cost is not.

3.1 Theory-Cost

In order to compute the number of different literals to be considered for evaluation, let us first consider the number of different variablizations of a single predicate P of arity m when the current clause has k old variables. Let this number be $v(m,k)$. To count the total number of distinct variablizations, it is convenient to count the number of distinct variablizations that use exactly j positions for old variables. We use $v(m,k,j)$ to represent this number. Since we require at least one old variable,

$$v(m,k) = \sum_{j=1}^{j=m} v(m,k,j).$$

Now $v(m,k,j)$ is the product of the number of ways of picking j positions, assigning old variables to them, and number of ways of assigning the new variables to the remaining positions. Clearly the total number of different ways filling j positions with old variables is $\binom{m}{j} * k^j$. There remains $m-j$ positions to fill with new variables. Since the name of a new variable is not important (i.e., they are dummy variables), we must count them carefully. For example, if X is an old variable and Y and Z are new variables then $between(Y,X,Z)$ and $between(Z,X,Y)$ are equivalent variablizations of *between*.

We define $new(i,j)$ to be the number of distinct (non-equivalent) ways of filling i positions with exactly j new variables. We note the following recurrence:[3]

$$new(i,j) = new(i-1,j-1) + j * new(i-1,j).$$

The boundary conditions are:

$$new(i,0) = 0$$
$$new(i,1) = 1$$
$$new(i,j) = 0, \quad if \quad i < j$$

[3]This recurrence was determined by Dan Hirschberg.

Table 3: Growth of $v(m,k)$

$v(m,k)$	Arity							
OLD variables	1	2	3	4	5	6	7	k
1	1	2	3	4	5	6	7	k
2	3	8	15	24	35	48	63	$(k+1)^2$
3	10	32	72	136	230	360	532	$\approx k^3$
4	37	136	357	784	1525	2712	4501	$\approx k^4$
5	151	622	1863	4684	10375	20826	38647	$\approx k^5$
6	674	3060	10278	29168	72810	163764	338030	$\approx k^6$

We define $new(i)$ to be the number of distinct ways of filling i positions with any number of new variables. By the definition of $new(i,j)$ we have

$$new(i) = \sum_{l=1}^{l=i} new(i,l).$$

Now the total number of variablizations of a predicate of arity m using j positions for old variables, $v(m,k,j)$, is:

$$v(m,j) = \left(\begin{array}{c} m \\ j \end{array} \right) * k^j * new(m-j).$$

Summing this function as j ranges from 1 to m will give the total number of ways of variablizing a predicate of arity m. Each non-equivalent predicate variablization determines two literals, one that is negated and another that is not. The information gain is computed for each such literal and one with the maximum gain is chosen. The rapid growth of $v(m,k)$ in both m and k is illustrated by Table 3.

For the full analysis, let $Pred(i)$ be the number of predicates of arity i. Let $Arity$ be the maximum arity of any predicate. As before, k is the number of old variables. The total number of literals to be considered is given by:

$$TheoryCost = 2 * \sum_{i=1}^{i=Arity} Pred(i) * v(i,k).$$

Because of the complexity of $v(m,k)$, this formula is somewhat difficult to use. Luckily, there are some simple bounds. Let Old be the maximum number of old variables. Then

$$Old^{Arity} \leq v(Arity, Old) \leq (Old + Arity - 1)^{Arity}.$$

Table 4: Definition of Member

$member(X, Y) \leftarrow component(X, Z, Y).$
$member(X, Y) \leftarrow component(A, B, Y), member(X, B).$

The first inequality is achieved by only using old variables. The second inequality is achieved by counting all distinct variablizations without regard to whether they are equivalent or not.

These inequalities allow us to generate a worst-case analysis for the theory cost. Let *Pred* be the number of predicates. To add a new predicate we may choose from one of *Pred* predicates. If the predicate has arity *Arity* (the worst case), then we must consider choosing *Arity* variables from *Old* old variables and $Arity - 1$ new variables. A simple upper bound on this cost is: $(Old + Arity - 1)^{Arity}$. Consequently, an upper bound of the theory cost is:

$$TheoryCost \leq 2 * Pred * (Old + Arity - 1)^{Arity} \quad (1)$$

One can make a number of qualitative inferences from this formula. In particular, it shows that additional predicates increase the cost (branching factor) by a linear amount, while increasing the arity of the predicates increases the size of the search space exponentially. Also, the amount of work increases exponentially with the number of distinct variables in the clause.

We have developed this analysis to measure the branching factor at any point in the search process. Now, let us turn this analysis around and use it to measure the complexity of learning some rule. Now, we are imagining that FOIL has learned some rule and we ask, approximately how many candidate rules were generated. In particular, imagine that we are learning concept of membership. This domain has 3 predicates, *null*, *member*, and *components*, with arities 1, 2, and 3, respectively. The definition of member is given in Table 4.

Let us concentrate our attention on learning the last conjunct of the second clause, which is clearly the point at which the branching factor is maximized. At this point the number of old variables is 4. There are 3 available predicates, one of each arity from 1 to 3. Using Table 3 we see that the number of extensions is $2 * (1 * 4 + 1 * 24 + 1 * 136)$.

There are two important points that should be noted. First, the branching factor is largely due to the predicates with the largest arity. Second, the branching factor of the last conjunct of the longest clause, measured in the number of distinct variables, is significantly larger than at other points in the search space.

Putting these observations together yields the following approximation for the $TheoryCost$ in learning a rule R. Let Var be the largest number of distinct variables in any clause of R, excluding the last conjunct. Let $MaxPred$ be the number of

predicates with largest arity $Arity$. Then an approximation of the total number of nodes generated to learn R is:

$$NodesSearch \approx 2*MaxPred*v(Arity, Var) \leq 2*MaxPred*(Var+Arity-1)^{Arity}.$$

Now that we know how may literals there are, we turn our attention to estimating cost of evaluating each literal.

3.2 Evaluation-Cost

In the previous section, we computed the number of different extensions of the current clause. Each such extension needs to be evaluated, and this is the main computational cost in running FOIL. This requires testing each literal on the current set of positive and negative tuples. By constructing tables of the number of solutions for each predicate with some of the values bound, it may be possible to reduce the cost of finding the number of solutions to problems of the sort $Pred(a, X, b, Y)$ to a single query. In any case, we suppose that this cost is a linear function of the number of tuples. This gives us our first estimate of the evaluation cost, namely:

$$EvaluationCost = TheoryCost * TupleSize. \quad (2)$$

As the literals in a clause are generated, the number of tuples can vary greatly. If the extension introduces no new variables, then the number of tuples will decrease, possibly by a very small amount. For example, it is possible that the extension will exclude one more negative tuple. In this case, an upper bound on the tuple size is simply the old tuple size. On the other hand, if the extension introduces new variables, then the number of tuples may increase dramatically. To estimate the size of the growth we will introduce a few new concepts.

First, we consider the case when no new variables are introduced by the literal. We define the *density* of a predicate to be the proportion of cases when the predicate is true. For example, suppose the domain is the integers from 1 to 10. Then the density of the $successor(X, Y)$ predicate is 9/100 and the density of $less(X, Y)$ is 40/100. The density of a negated predicate is defined analogously. If a literal introduces no new variables, then the tuple size must decrease as it consists of the subset of the current tuples which satisfy the literal. In this case, we expect that the new tuple size will be the density of the predicate times the old tuple size.

Now we consider the case where the literal introduces new variables. We define the *power* of a predicate to be the maximum number of solutions of the predicate when one variable is bound. For the predicate $less(X, Y)$ on the domain of integers from 1 to 10, the power is 9, which is achieved by $less(1, Y)$ and by $less(X, 10)$. Similarly, the power of the *successor* predicate is 1. Since negative literals do not introduce new variables, we define their power to be 1. The power of a predicate limits the amount of growth in the tuple size, since $NewTupleSize \leq OldTupleSize * PowerOfPredicate$.

We now use the notion of *Power* to get an upper bound on the growth of the tuple size. Let P_i, for $i = 1$ to k be the literals in the body of a clause. Define $growth(P_i) = 1$ if P_i uses only old variables and $growth(P_i) = Power(P_i)$ if P_i uses new variables. Then a conservative estimate for the tuple size is

$$TupleSize \leq \prod_{i=1}^{k} Growth(P_i). \tag{3}$$

We can achieve a more reasonable estimate for the expected tuple size. Define the *AveragePower* of a predicate to be the average number of solutions of the predicate when one variable is bound. Since neither $successor(X, 1)$ or $successor(10, Y)$ have any solutions in the domain, the AveragePower of the *successor* predicate is 18/20. Over the same domain, the predicate *less* has AveragePower 4.5. The importance of the power of a predicate is that, in the worst case, the number of tuples can increase by no more than the power of the predicate. It is reasonable to expect that the tuple size would grow by the AveragePower, rather than Power.

Define he $AverageGrowth(P_i)$ to be density of P_i if P_i uses only old variables and $AveragePower(P_i)$ if P_i uses new variables. This yields the following approximation for the tuple size:

$$TupleSize \approx \prod_{i=1}^{k} AverageGrowth(P_i). \tag{4}$$

The importance of these estimates indicates that in order to reduce the evaluation cost, we should prefer predicates that have low average power and low density. A predicate like *successor*, which has power of 1, is guaranteed not to increase the tuple size. Of course, we always prefer predicates that most increase the information gain. In Table 12, we illustrate the accuracy of these approximations.

Our general conclusions from this analysis are that the number of literals to add to the end of a clause grows exponentially with the arity of the predicates and the number of variables, which is likely to be proportional to clause length. The number of examples can also grow, but this growth is bounded by the power of the predicate. Consequently, one might choose predicates with low power when representing a domain.

In the subsequent sections, we will show how that by adding knowledge, we can reduce, sometimes dramatically, these costs. Somewhat surprisingly, this analysis will also show that sometimes large amounts of knowledge will have very little effect on reducing the search space.

4 FOCL

FOCL extends FOIL in a variety of ways. Each of these extensions affects only how FOCL selects literals to test while extending a (possibly empty) clause under construction. These extensions allow FOCL to take advantage of domain knowledge to guide the learning process. One class of extensions allows FOCL to use

constraints to limit the search space. A second type of extensions allows FOCL to use intensionally defined predicates (i.e., predicates defined by a rule instead of a collection of examples) in a manner similar to the extensionally defined predicates in FOIL. A collection of intentionally defined predicates is identical to the domain theory of EBL (Mitchell, Keller, & Kedar-Cabelli, 1986). A final extension allows FOCL to accept as input a partial, possibly incorrect rule that is an initial approximation of the predicate to be learned. If this rule is defined in terms of extensionally defined predicates, it is analogous to a partial concept definition constructed by an incremental inductive learning system. If this rule is defined in terms of intensionally defined predicates, it is analogous to the target concept of EBL. Indeed, when we discuss explanation-based extensions to FOCL, we will use the terms "non-operational" and "intensionally defined" as synonyms. Similarly, the extensionally defined predicates correspond to the observable facts (or the operational predicates) of EBL. The goal of FOCL, like FOIL, is to create a rule (i.e., a set of clauses) in terms of the extensionally defined predicates, that covers all of the positive examples and none of the negative examples.

In the following sections, we describe these extensions in more detail and evaluate the effect of each extension on the number of literals tested by FOCL or the accuracy of FOCL. To illustrate these extensions, we use two domains. The first domain, that of learning the *member* predicate, illustrates how a simple recursive concept can be learned. FOIL is provided with positive and negative examples of the *member* predicate (e.g., $member(b, [a, b, c])$ $notmember(a, [b, c])$ and the *component* predicate (e.g., $component(a, [b, c], [a, b, c])$ and learns the correct recursive definition for *member*, as given in Table 4.

The second domain is much more complicated and was introduced by Muggleton *et al.* (1989). This domain suggests that FOCL can handle moderately-sized realistic domains. Several hundred examples are used to build a concept description that varies from four to eleven clauses, depending upon the extensional predicates that are provided. The predicate or concept to be learned is $illegal(A, B, C, D, E, F)$. That is true if a chess board containing a white king and rook and black king is in an illegal state. A state is illegal if either king is in check or more than one piece occupies the same space. A and B are the position of the white king (rank and file), C and D are the white rook's position, and E and F are the black king's position. The ranks and files are represented by a number between 1 and 8. In this example, the operational predicates used are $between(X, Y, Z)$ (the value of Y is between the values of X and Z), $adjacent(X, Y)$ (the value of X is either one greater or one less than the value of Y) and $equal(X, Y)$ (the values of X and Y are equal).

Where appropriate, we present two sorts of experiments with this domain. First, we present experiments using a large number of examples that enables FOCL to learn a concept description that is extremely accurate ($> 99\%$) and we measure the effect of the knowledge on the size of the hypothesis space searched. Second, we

present experiments using a smaller number of examples and evaluate the impact of the knowledge on the accuracy of the rule learned.

4.1 Zero Knowledge Differences in FOIL and FOCL

The goal of our analysis and experimentation is to gain an understanding of the impact of each type of knowledge on acquiring Horn clause theories. However even when FOCL is provided with no knowledge, it has some differences with FOIL that we should mention.

As noted in Section 3, the theory-cost and the evaluation-cost grows exponentially in the number of distinct variables. Consequently, to lessen this cost, in FOCL, we have introduced an iterative widening search strategy[4] that is analogous to iterative depth-first search (Korf, 1985). FOCL first attempts to learn a clause by introducing no free variables. If that fails (because a situation is encountered where no variablization of any predicate has positive gain), an attempt is made to learn the clause by allowing an additional free variable. On each failure, an additional free variable is allowed until the number of free variables exceed the maximum arity of any predicate. There is a small cost for using iterative widening search when it is not needed. However, when iterative widening reduces search, there is a major benefit.

Additionally, there are three features of FOIL that we do not consider in this paper. First, FOIL contains a limited form of backtracking to allow it to solve some problems that cannot be solved with hill climbing alone. It is difficult to estimate how often this backtracking is needed. All of the examples in this paper can be solved without backtracking. Second, FOIL contains a branch-and-bound pruning heuristic that allows it to avoid testing the variablizations of some predicates. This operates by computing the maximum possible information gain of a predicate variablization from the information gain of a more general variablization (i.e., a variablization can be more specific than another by replacing a new variable with an old variable). It is difficult to analyze the impact of the pruning heuristic on the number of literals tested. In the worst case, it will have no impact, and it will never affect the accuracy of the hypothesis. Since this heuristic is not compatible with the iterative widening search, we do not make use of it. Finally, FOIL contains an information-based stopping criteria that allows it to learn from noisy data. We do not consider noisy data in our analysis or experiments.

Now we will describe the extensions and modifications of FOIL that permit various forms of background knowledge to be exploited. We also evaluate the benefit that these extensions have. After we have considered each of these extensions separately, we present the complete FOCL algorithm.

[4]It is also similar to the iterative broadening technique (Ginsberg & Harvey, 1990) where heuristics are used to order the nodes expanded. In our case the heuristic is to favor nodes with few new variables. Their analysis was for constant branching factor and the success of the method relied on having enough goal nodes. In our case, the branching factor is dependent on the extension chosen.

4.2 Single Argument Constraints

Type constraints provide a useful and inexpensive way of incorporating a simple form of background knowledge. FOCL can easily take advantage of typing information.[5] Typing is implemented by associating a type for each argument of a predicate. For example, the predicate $illegal(A, B, C, D, E, F)$ has a type definition of $illegal(rank, file, rank, file, rank, file)$. A type can then be associated with a variable the first time it is used in a clause and all other uses of that variable in the clause must be consistent with that type.

Introducing typing may require introducing additional predicates. In the *illegal* example, the predicate *adjacent* is overloaded in that it can compare ranks or compare files. However, it should never be used to compare ranks to files. Therefore, we add the predicate $adjacent\_rank(X, Y)$ with the type $adjacent\_rank(rank, rank)$. Similarly, $adjacent\_file(A, B)$ is used to compare files.

Typing reduces the search space by avoiding testing literals where the types of old variables conflict with the usage of these variables as arguments to a predicate. More precisely, let us assume that a domain has T types and, in the best case, these types are distributed equally among the variables. Then, with typing, theory-cost is reduced to approximately $TheoryCost = Pred * (Var + Max)/T^{Max}$, a savings of T^{Max}. This shows that, in the best case, typing can reduce the search space by an exponential amount. In practice, the reduction, though significant, is less than the best case.

In the chess domain, typing information was used to ensure that the predicates *between*, *equal*, and *adjacent* were only applied to either all ranks or all files. The benefit of typing is illustrated by the fact that FOCL using typing tests 3240 literals and 242,982 tuples as compared to 10,366 literals and 820,030 tuples for FOCL without typing when learning *illegal* from 641 randomly selected positive and negative training examples, of which 233 were positive and 408 were negative.

In addition to reducing the size of the search space explored, typing can also improve the accuracy of the hypothesis produced. The effect of typing on the hill-climbing search is to eliminate some literals that may (coincidentally) have the maximum information gain. For example, in the chess domain, it can occur that a literal that violates the typing constraints has the maximum information gain (e.g., the rank of the white king is equal to the file of the black king). Typing prevents FOCL from considering these literals. We ran 10 trials of FOCL both with typing and without typing on 10, 20, 40, 60, 80, 100, 150, and 200 randomly selected training examples and measured the accuracy at each point on 1000 randomly selected training examples. Figure 1 shows the mean accuracy plotted as a function of the number of training examples.

[5] Quinlan (1990) mentions how type constraints may be used (in combination with the closed-world assumption) to generate negative examples of the predicate to be learned from the positive examples. However, type constraints are not used to eliminate literals from consideration.

========== Figure 1 ==========

With a small number of examples, typing improves the accuracy of the resulting hypotheses produced by FOCL. As the number of examples increases, the effect of typing on accuracy is reduced. This occurs because it is unlikely in a larger training set for a predicate variablization that does not obey the typing restriction to have the maximum information gain. However, typing is still useful since it is an inexpensive way to reduce the number of literals tested.

4.3 Multiple Arguments Constraints

A second type of a constraint involves inter-argument constraints, the relationship between the arguments of a predicate. For example, $equal(X, X)$ is trivially true and $between(X, X, Y)$ is trivially false. Such expressions should not play a part in a concept definition and, therefore, it is wasteful to test hypotheses including these literals.

One case we have implemented indicates that for some predicates it is necessary for all of the variables in one literal to differ. The definition of each predicate may contain a *unique_variables* declaration to indicate whether or not this predicate requires all variables to differ. Providing such constraints on literals when FOCL learns *illegal* further reduces the size of the hypothesis space explored. Like typing, inter-argument constraints reduce the number of variablized literals that must be tested.

The value of inter-argument constraints is illustrated by the fact that FOCL, using typing and inter-argument constraints, tests 1296 literals and 109,350 examples as compared to 3240 literals and 242,982 examples for FOCL using only typing when learning *illegal* from 641 randomly generated training examples.

This unique variables inter-argument constraint does not affect the accuracy of the resulting hypotheses. A trivially true or trivially false predicate cannot have positive information gain. Nonetheless, like variable typing, it is an effective constraint for reducing the number of literals that are tested by the inductive component of FOCL. These constraints reduce CPU time as well as the theory-cost and evaluation-cost. For example, without these constraints, FOCL took 866 CPU seconds on an Apple Macintosh II computer running Common Lisp. With these constraints, 179 CPU seconds were consumed during learning *illegal*.

We have also implemented a second inter-argument constraint: commutativity. For example, it is not necessary to test $adjacent\_rank(Y, X)$ since this has the same meaning and information gain as $adjacent\_rank(X, Y)$. Binary predicates may be declared to be commutative, reducing the number of literals explored. In the *illegal* example, *adjacent* and *equal* are commutative. Adding this knowledge reduces the number of variablization of commutative predicates by half. Therefore, a total of 711 literals were tested with this additional knowledge. Note that commutativity does not affect accuracy, but merely avoids testing equivalent variablizations of the same literal.

4.4 Operational Initial Rules

In the next sections, we consider ways in which background knowledge can improve upon inductive learning. First, we will consider the case where the background knowledge is a (possibly incorrect) partial, operational rule that approximates the concept to be learned. For the subsequent discussion we regard an operational predicate as one that is given extensionally. If a predicate is defined by other predicates, we say the definition is non-operational. Such an initial rule might be provided by a teacher, or, in an incremental learning system (e.g., Widmer, 1990), learned from an initial subset of the examples.

The extension to FOCL to use a partial, operational Horn clause rule is straightforward. In FOIL, the information gain of a literal is computed as a function of the original and extended positive and negative tuples covered by the literal. A clause is merely a conjunction of literals. Therefore, the information gain of a clause is simply a function of the number of tuples covered by the conjunction of literals. When deciding to add a new literal, FOCL computes the information gain of each clause in the initial concept. In *theory-mode*, if any clause has positive information gain, the conjunction of literals is added to the clause under construction. Theory-mode has a bias in favor of the initial rule. In *information-mode*, FOCL compares the information-gain of each clause to that of all variablized extensionally defined predicates. The literal (or conjunction of literals) with the maximum gain is added to the end of the (possibly null) current clause. If the current clause covers some negative tuples, additional literals are added to rule out the negative tuples.

The analysis of the complexity of FOIL provides insight into the benefit of including an operational partial rule for the predicate to be learned. In general, search in FOIL is dominated by the last literal of the clause with the largest number of variables. This means that a partial rule that is nearly complete, but omits the last literal of the clause with the largest number of distinct variables, reduces the search by only a negligible amount.

The following experiments support this analysis. We gave FOCL three partial definitions of the member function, namely:

1. $member(X, Y) \leftarrow component(X, Z, Y).$
2. $member(X, Y) \leftarrow component(X, Z, Y).$
   $member(X, Y) \leftarrow component(A, B, Y).$
3. $member(X, Y) \leftarrow component(X, Y, Z).$

The first two definitions are partial and correct. The second clause of the second partial definition must be extended by adding an additional literal. The last partial definition is incorrect. FOCL tests 268, 228 and 308 literals and considers 20,140, 12,167 and 23,358 tuples with the above partial concept definitions as compared to FOIL's 308 literals and 23,057 tuples. Note that the correct partial definitions given do not significantly reduce the number of literals tested because the majority

of the work is needed to add the last literal to the last clause of member. The incorrect partial definition does not increase the number of literals tested but slightly increases the number of tuples tested since it must check the possibility that the advice may have some advantage.

In general, a partial operational concept definition reduces search in FOCL since FOCL saves the work needed to generate this partial definition. However, since this search is dominated by the last literal of the clause with the largest number of distinct variables, a partial definition that does not contain this clause does not save a significant amount of work.

4.5 Non-Operational Predicates

Next, we consider domain theories using non-operational predicates, i.e., ones defined in terms of operational and other non-operational predicates. Systems such as CIGOL (Muggleton & Buntine, 1988) make use of (or invent) background knowledge of this form. For example, if an operational definition of the predicate $between(X, Y, Z)$ is not provided, it could be defined in terms of the operational predicate *less_than* by: $between(X, Y, Z) \leftarrow less\_than(X, Y), less\_than(Y, Z)$.

One advantage of the non-operational predicates is illustrated by the fact that $between(X, Y, Z)$ may have positive information gain, while $less\_than(X, Y)$ and $less\_than(Y, Z)$ may have negative gain. Therefore, FOIL's hill-climbing search may not learn a concept that involves $less\_than(X, Y), less\_than(Y, Z)$. More generally, non-operational predicates allow the hill-climbing search to take some larger steps that can allow the hill climber to solve problems that cannot be solved with smaller steps.

Note that it would be computationally prohibitive to consider all conjunctions of length two of the operational predicates. In general, this would more than square the theory-cost. Non-operational predicates provide information on what particular combinations of operational predicates may be useful and allow FOCL to simulate a selective look-ahead.

Non-operational predicates are evaluated in the same manner as operational predicates in FOCL. The information gain of all variablizations of non-operational predicates is computed in a manner similar to that used by FOIL with operational predicates. Computing the information gain of a non-operational literal requires counting the number of positive and negative tuples (and extensions of these tuples if the variablization includes new variables) covered by the literal.[6] If the literal with the most gain is non-operational, then the literal is operationalized and the operational definition is added to the clause under construction. Note that, unlike operational predicates, the computation of the information gain of non-operational predicates involves a Prolog proof.

[6]The Prolog predicate *setof* can be used to find all extensions of a tuple. For example, if the current tuple is (1 2 3 4 5 6) (corresponding to the variables A, B, C, D, E, and F) then the extensions of this tuple for the literal $between(A, G, E)$ requires finding all bindings of G such that $between(1, G, 5)$ is true.

Table 5: Operationalization

```
Procedure: Operationalize(Predicate, Pos, Neg)
Initialize ClauseBody to the empty set.
For each clause in the definition of Predicate
  compute_gain(clause, Pos, Neg).
For the clause with the maximum gain,
  for each literal T in the clause,
    if T is operational, add T to ClauseBody
    else add operationalize(T, Pos, Neg) to ClauseBody.
```

The operationalization process in FOCL differs from that of EBL in that it is guided by an information gain metric over a set of both positive and negative examples rather than by the proof of a single positive example. As in EBL, the operational definition for a predicate may specialize the predicate if the domain theory is disjunctive (i.e., if there are multiple clauses for any non-operational predicate). In EBL, the predicates that are the leaves of the proof tree of the single training example are used as the operational definition. In FOCL, the information gain metric is used to determine how to expand a proof tree, as in Table 5.

The compute_gain function uses Prolog to prove a clause (i.e., a conjunction of operational and non-operational literals). The operationalization process uses the information gain metric to select which clause of non-operational rule should be expanded. The result of this process is that an operational specialization of a non-operational literal is selected that covers many positive tuples and few negative tuples.

Due to its reliance on hill-climbing search, FOIL and FOCL are unable to learn a completely correct definition of *illegal* using only *less_than*, *equal* and *adjacent*. When FOCL is also given a non-operational definition of *between* in terms of *less_than*, it finds a completely correct definition in terms of the operational predicates *less_than*, *equal* and *adjacent*.

A disadvantage of using non-operational predicates in this manner is that each additional non-operational predicate, particularly those with many arguments, increases the search space. This has the undesirable consequence that the more one knows, the slower one learns. This became obvious when we added rules from a domain theory of chess to FOCL. These rules indicate facts such as a king is in check if there is an opposing rook in the same file as the king and there is not another piece between the rook and king. Table 6 contains a definition of these predicates for FOCL.

With this domain theory, FOCL tested 3063 literals and 283,602 tuples to find an operational concept definition as opposed to 1296 literals and 109,350 tuples

when just the operational predicates are searched when learning *illegal* under the same conditions reported previously. This experimental finding agrees with the analysis of FOIL presented in Section 3. In particular, the number of predicates increased, so the number of literals tested increased.

A positive effect of a domain theory is that it might provide the right predicates to allow a hill-climbing search to find the concept description. On the negative side, it increases the search space and can decrease the accuracy. This can occur if the predicates present in the domain theory are irrelevant to the task. A variablization of these irrelevant predicates may have the maximum information gain, and be used as literals in a clause. This problem is not limited to just non-operational predicates.

To demonstrate this, we added the irrelevant predicates $odd(X)$, $prime(X)$, $successor(X,Y)$, $plus(X,Y,Z)$, $times(X,Y,Z)$, $square(X,Y)$, $cube(X,Y)$, and $greater(X,Y)$. We ran 10 trials of FOCL, both with extra irrelevant predicates and without extra irrelevant predicates on 10, 20, 40, 60, 80, 100, 150, and 200 randomly selected training examples. Then we measured the accuracy on 1000 randomly selected training examples. Figure 2 shows the mean accuracy plotted as a function of the number of training examples. When learning with extra irrelevant predicates, especially *greater* and *successor*, FOCL learned concepts that were not as accurate on test data. The original operational predicates, *equal*, *between* and *adjacent* were carefully chosen because these concepts are useful in learning about rooks attacking kings, kings attacking kings, and blocking a rook from attacking a king. When less care is taken in selecting predicates, or if the relevant predicates cannot be determined beforehand, more data is needed to create accurate hypotheses. In this manner, the available predicates in a Horn clause learner are analogous to the attributes used by a propositional learning program. In summary, irrelevant predicates can increase the amount of work, by a linear amount, and increase the number of training examples required to achieve a given accuracy.

========== Figure 2 ==========

4.6 Non-Operational Initial Rules

In the previous section, we pointed out how adding background knowledge in the form of a domain theory can increase the ability of FOCL to find solutions. However, increasing the size of the domain theory may increase the search space explored by the learning program and decrease the accuracy of the resulting hypothesis. In explanation-based learning, the search for a concept definition is facilitated by providing the learning system with an overly general concept description or target concept. The target concept is assumed to be a correct, non-operational definition of the concept to be learned and the domain theory is assumed to be correct. In FOCL, we relax the assumptions that the target concept and the domain theory are correct. Because FOCL makes use of multiple training examples, it has the potential for learning a correct rule in spite of these inaccuracies.

Table 6: Partial Domain Theory for Chess

Predicate to be learned: $illegal(A, B, C, D, E, F)$
$Type$: $(rank\ file\ rank\ file\ rank\ file)$
Pos : (5 3 1 8 1 6)(3 7 5 6 1 6) ...
Neg : (3 8 6 1 8 5)(8 6 4 1 1 8) ...

NonOperational Predicates:
$same\_loc(R1, F1, R2, F2) \leftarrow equal\_rank(R1, R2),\ equal\_file(F1, F2).$
$type$: $(rank\ file\ rank\ file),\ unique\_variables.$
$king\_attack\_king(R1, F1, R2, F2) \leftarrow adjacent\_rank(R1, R2),\ adjacent\_file(F1, F2)$
$king\_attack\_king(R1, F1, R2, F2) \leftarrow adjacent\_rank(R1, R2),\ equal\_file(F1, F2)$
$king\_attack\_king(R1, F1, R2, F2) \leftarrow equal\_rank(R1, R2),\ adjacent\_file(F1, F2)$
$type$: $(rank\ file\ rank\ file),\ unique\_variables.$
$rook\_attack\_king(R1, F1, R2, F2, R3, F3) \leftarrow equal\_rank(R2, R3),$
$king\_not\_between\_file(R1, F1, R2, F2, F3).$
$rook\_attack\_king(R1, F1, R2, F2, F3, R3) \leftarrow equal\_file(F2, F3),$
$king\_not\_between\_rank(R1, F1, R2, F2, R3).$
$type$: $(rank\ file\ rank\ file\ rank\ file),\ unique\_variables.$
$king\_not\_between\_file(R1, F1, R2, F2, F3) \leftarrow not(equal\_rank(R1, R2)).$
$king\_not\_between\_file(R1, F1, R2, F2, F3) \leftarrow equal\_rank(R1, R2),$
$not(between\_file(F2, F1, F3)).$
$type$: $(rank, file, rank, file, file),\ unique\_variables.$
$king\_not\_between\_rank(R1, F1, R2, F2, R3) \leftarrow not(equal\_file(F1, F2)).$
$king\_not\_between\_rank(R1, F1, R2, F2, R3) \leftarrow equal\_file(F1, F2),$
$not(between\_rank(R2, R1, R3)).$
$type$: $(rank, file, rank, file, rank),\ unique\_variables.$

Operational Predicates:
$between\_rank(rank, rank, rank),\ unique\_variables.$
Pos : (1 2 3)(1 2 4)(1 2 4)...(2 3 4),(2 3 5) ... (6 7 8)
$equal\_rank(rank, rank),\ unique\_variables.$
Pos : (1 1)(2 2)(3 3)(4 4)(5 5)(6 6)(7 7)(8 8)
$adjacent\_rank(rank, rank),\ unique\_variables.$
Pos : (1 2)(2 1)(2 3)(3 2)(3 4)...(7 8)

Table 7: Chess Target Concept

$illegal(R1, F1, R2, F2, R3, F3) \leftarrow same\_loc(R1, F1, R2, F2).$
$illegal(R1, F1, R2, F2, R3, F3) \leftarrow same\_loc(R1, F1, R3, F3).$
$illegal(R1, F1, R2, F2, R3, F3) \leftarrow same\_loc(R2, F2, R3, F3).$
$illegal(R1, F1, R2, F2, R3, F3) \leftarrow king\_attack\_king(R1, F1, R3, F3).$
$illegal(R1, F1, R2, F2, R3, F3) \leftarrow$
$rook\_attack\_king(R1, F1, R2, F2, R3, F3).$

When a non-operational initial rule is provided to FOCL, it is treated in a manner similar to an operational rule (i.e., a rule using just extensionally defined predicates). In particular, it is possible to compute the information gain of a conjunction of extensionally and intensionally defined literals, by using a Prolog style proof process to determine which examples (and extended examples) are covered by each clause of the initial rule. In theory-mode, FOCL operationalizes a clause provided that it has positive information gain. In information-mode, it is compared to the induced literal with the maximum information gain.[7] If a clause of the initial rule is non-operational and has maximum gain, it is operationalized in the manner described in Section 4.4.

Table 7 shows a representation of a initial non-operational rule (target concept) for *illegal*.

In theory-mode, when FOCL is provided with a correct non-operational target concept and the domain theory of the previous section, it finds a correct operational definition of *illegal* by testing 72 literals. In contrast, with the correct domain theory, but no target concept, FOCL tests 3063 literals. The reason for this savings is that with no target concept, FOCL must test every variablization of every predicate. When provided with a target concept, the proof structure determines which variablizations are tested. For example, only 3 variablizations of *same_loc* are tested. At this point in the computation, there are 6 variables and *same_loc* has arity 3. Consequently, by Table 3 there are 10,278 different variablizations. Without the target concept, the constraint of typing, unique variables, and iterative widening search reduces this number to 36. Similarly, only one variablization of *rook_attack_king* is tried during operationalization instead of the 163,764 needed without any constraints.

[7] In our experience, the mode does not have a significant effect on the accuracy of the hypotheses produced by FOCL unless the domain theory is extremely inaccurate (i.e., less than 60% accurate). When the domain theory is more accurate, theory-mode results in less search than information-mode. When the domain theory is less accurate, theory-mode results in more search because the operationalized concepts tend to be overly specialized and more clauses are needed to cover the training examples.

Note that typing, unique variables, and iterative widening are not needed by the analytic learning component, since the domain theory and target concept control the selection of predicate variablizations. A good domain theory will not violate typing, use predicates trivially, or introduce unnecessary new variables.

In addition to reducing the search space, a correct target concept and domain theory will, of course, improve the accuracy. We ran 10 trials of FOCL, both with a correct domain theory and target concept and without this knowledge on 10, 20, 40, 60, 80, 100, 150, and 200 randomly selected training examples, and measured the accuracy at each point on 1000 randomly selected training examples. Figure 3 shows the mean accuracy plotted as a function of the number of training examples. As expected, the correct target concept and domain theory improve the accuracy of the resulting hypothesis.

========== Figure 3 ==========

In the current implementation of FOCL, a domain theory and target concept does reduce the CPU time, although not as dramatically as typing and inter-argument constraints. A total of 512 CPU seconds on an Apple Macintosh II were consumed using purely explanation-based learning as compared to 866 seconds using induction with no constraints. The current implementation makes use of a backward chaining rule interpreter implemented in Common Lisp that runs at approximately 500 logical inferences per second. An analysis of the time consumed indicates that a major portion of the time is spent "proving" operational predicates. For example, proving $between(6, 7, 8)$ requires attempting to unify this literal with 112 facts. Some sort of indexing could considerably speed up the explanation-based portion of FOCL. In the inductive portion of FOCL, facts are stored in hash tables that allow constant time retrieval.

4.7 Learning Non-Operational Concept Definitions

A relatively minor modification to FOCL also allows it to learn non-operational concept definitions that can be used as target concepts for EBL(Pazzani & Brunk, 1990). Presented with a domain theory, but no target concept, FOCL searches for all variablizations of all operational and non-operational predicates. As discussed in Section 4.4, FOCL then operationalizes any non-operational predicate variablization. By simply eliminating this operationalization process, FOCL can induce a non-operational concept definition. For example, FOCL learns a non-operational definition of *illegal* that is identical to that in Table 6.

4.8 Summary of FOCL

Now that we've explained each of the individual pieces of FOCL, we show how they fit together in Table 9. This is a high level design that emphasizes the differences with FOIL without going into too many of the details. FOCL extends FOIL in several ways. First, there are constraints on the inductive process so that not all variablizations of a predicate need to be checked. Second, FOCL can compute the information gain of intensionally defined predicates (in addition to extensionally de-

Table 8: FOCL Specification

Given:

1. The name of a predicate of known arity.
2. A set of positive tuples.
3. A set of negative tuples.
4. A set of extensionally defined predicates.
5. (optionally) A set of intensionally defined predicates.
6. (optionally) A set of constraints (e.g., typing) on the intensional and extensional predicates.
7. (optionally) An initial (operational or non-operational) rule.

Create: A rule in terms of the extensional predicates such that no clause covers any negative examples and some clause covers every positive example.

fined predicates). Third, FOCL can operationalize intensionally defined predicates by finding an operationalization specialization that covers many positive and few negative examples. Fourth, FOCL can compute the information gain of an initial (operational or non-operational) rule for the concept to be learned and decide to use this in favor of induction. In this view, the value of an initial rule (i.e., target concept) is that it indicates the variablizations of a non-operationally predicate that are likely to be useful. Table 9 shows an outline of the FOCL algorithm. For simplicity, the algorithm is specialized to FOCL in theory-mode and we do not consider the case where FOCL is instructed not to operationalize intensionally defined predicates.[8]

The ability of FOCL to deal with incomplete and incorrect domain theories comes from having a uniform information gain metric that is applied to both inductively formed or analytically formed literals. The only difference between inductively formed and analytically formed literals is that the search for an analytically formed literal is more directed. The decision about whether to use inductive or explanation-based techniques to extend a clause is based on the usefulness in producing an accurate hypothesis, as measured by the information metric.

5 Incorrect and Incomplete Domain Theories

FOCL is capable of utilizing incorrect and incomplete domain theories. FOCL tolerates such theories because the literals proposed by analytic methods are tested by an information-based metric to make sure they have positive gain (or the maximum gain). If an analytical extension is not selected, then FOCL selects literals inductively.

[8]Note that to avoid conjoining two different clauses of an initial rule, the InitialRule is set to empty if it is used to add literals to a clause. It is reset on the start of a new clause, so that other clauses may be used.

Table 9: Design of FOCL

```
Let P be the predicate to be learned.
Let POS be the positive tuples.
Let NEG be the negative tuples.
Let IR in the initial rule.
Let Body be empty.

Until POS is empty
  Call LearnClauseBody.
  Remove from POS those tuples covered by Body.
  Set Body to empty.

Procedure LearnClauseBody:
    If a ClauseBody of IR has positive gain,
      Select it, (*1)
      Operationalize it (if necessary), (*3)
      Conjoin it with Body,
      Update POS and NEG,
      Call ExtendBody (*2)
    Else
      Choose best literal,
      Operationalize it (if necessary),(* 3)
      Conjoin result with body,
      Update POS and NEG,
      Call LearnClauseBody.

Procedure ExtendBody:
  While NEG is non-empty
    Choose best literal. (*3)
    Operationalize it.
    Conjoin it with Body.
    Update POS and NEG.

Notes:
  *1: takes advantage of good priori clauses.
  *2: allows correction of old clause bodies.
  *3: allows use of non-operational predicates.
```

Table 10: Domain Errors

Deleted Clause:
$rook\_attack\_king(R1, F1, R2, F2, R3, F3) \leftarrow equal\_rank(R2, R3),$
$king\_not\_between\_file(R1, F1, R2, F2, F3).$

Added Clause:
king_attack_king(R1,F1,R2,F2)←knight_move(R1,F1,R2,F2).

Deleted Literal:
Changed: $king\_attack\_king(R1, F1, R2, F2) \leftarrow$
$equal\_rank(R1, R2), adjacent\_file(F1, F2)$
To: *king_attack_king(R1,F1,R2,F2)←equal_rank(R1,R2).*

Added Literal:
illegal(R1,F1,R2,F2,R3,F3)←same_loc(R1,F1,R2,F2),
adjacent_rank(F1,F3).

To illustrate how FOCL learns in spite of incorrect domain theories, we simultaneously introduced four errors into the correct domain theory for *illegal*. The errors are indicated in Table 10.

These errors correspond to indicating that rooks may only attack in files, kings may also move like a knight, kings may attack anywhere in the same rank, and two pieces may occupy the same square provided the black king is in an adjacent rank. With these four errors, the domain theory correctly classified 76.2% examples when tested on 10,000 training examples. Appendix I provides an edited trace of FOCL operating with this domain theory (in theory-mode). FOCL tests 705 literals to generate this definition. Analysis of the definition, confirmed by testing on 10,000 training instances, indicates that the concept acquired is 100% correct.

To create an operational definition for *illegal*, FOCL first computes the information gain of each given clause for *illegal*. In this case, *rook_attack_king* has the highest information gain, and is operationalized. Since there is only one clause for *rook_attack_king* in this incorrect domain theory, it is selected for operationalization. This clause uses one operational predicate *equal_file* and one non-operational, *king_not_between_rank*. There are two clauses for *king_not_between_rank* and the clause with the maximum information gain is operational. Therefore, the first clause for *illegal* is created entirely with explanation-based methods in FOCL. This clause indicates that a chess board is in an illegal state if the white rook and black king are in the same file, and the white king is not in the same file. The positive examples that are satisfied by this clause are removed and the same process is repeated. Note

that since the set of positive tuples is reduced, the information gain of the clauses for *illegal* is different when learning the first and second clause.

The second clause operationalizes *king_attack_king*. The most common way for this to occur is for the two kings to be in adjacent ranks and adjacent files. The third clause also operationalizes *king_attack_king*. This time the operationalization indicates that the kings are in the same file and adjacent ranks. If the domain theory was correct and complete, this process would be repeated until all positive examples are covered by at least one operational clause. Clauses would be created in a greedy manner by selecting the operationalization that covers the most positive examples (i.e., if no operationalization covers any negative examples, then the operationalization that covers the most positive examples has the highest information gain). However, since an incorrect domain theory will misclassify some negative examples, and an operationalization of an incomplete theory will fail to cover some positive examples, it is also necessary to use the inductive component. Clause 4 provides one example where both the inductive and the explanation-based components are needed.

In Clause 4, FOCL operationalizes *king_attack_king* again. However, the clause with the maximum information gain is the clause with the deleted literal. This indicates that a king attacks a king if they are in the same rank. Because some negative examples are covered by this clause, the clause is extended. FOCL first tries to operationalize another conjunct of the target concept. However, in this case, there is no conjunct since the target concept is a single literal. Next, FOCL uses the inductive techniques of FOIL to extend this clause. It computes the information gain of every variablization of every operational predicate and selects the literal with the maximum gain, $adjacent\_file(F3, F1)$. This excludes all negative examples and the clause added indicates that a chess board is in an illegal state if the kings are in the same rank and adjacent files.

Clause 5 is learned by operationalizing *same_loc*. It indicates that a chess board is in an illegal state if the two kings are on the same square. Clause 6 is learned by operationalizing *rook_attack_king*. It covers the case that is less common than the first clause. In this case, the white rook and black king are in the same file. The white king is in this file, but not between the rook and black king.

Clause 7 is learned entirely by inductive techniques. It indicates that a chess board is illegal if the white rook and black king are in the same rank, and the white king is not in that rank. Note that after the first literal is added inductively, FOCL again tries to operationalize the target concept. In this case, no operationalization of the target concept has positive information gain when extending this clause. However, it can occur that the first literal of a clause is learned inductively, and some of the remaining literals are learned via explanation-based techniques.

After the set of positive examples that matches this clause is removed, a clause of the target concept now has positive information gain (*king_attack_king*). Once

that clauses had at least 2 terms, a .25 probability of at least 3, etc. This modification will cause the rule to make errors on negative training examples.

We train FOCL on a large number of training examples, and in all cases, the resulting hypothesis is greater than 99% accurate. In each case, we measure the amount of search that is required to create a hypothesis.

In the first set of experiments, each perturbation operator was applied individually. Figure 5 plots the accuracy of the resulting domain theory (averaged over 10 trials on either the positive or the negative training tuples as appropriate according to the type of modification) and the number of literals tested by FOCL in theory-mode for each operation as a function of the number of modifications to the domain theory. (There were fewer than 10 modifications possible for deleting clauses or literals.) Note that FOCL is able to exploit extremely inaccurate domain theories to constrain the search for a concept definition.

========== Figure 5 ==========

The easiest problem for FOCL occurs when additional clauses are added to the domain theory. This problem can be solved entirely by explanation-based means. A subset of the possible operationalizations of the target concept are chosen in a greedy manner to cover the positive examples and exclude the negative examples. The more difficult problems for FOCL occur when the inductive component of FOCL is required to make up for an inadequate domain. Induction is needed when no subset of the possible operationalizations of the domain theory will result in a correct hypothesis.

We also ran experiments in which all of the above modifications were performed simultaneously on the domain theory, yielding a domain theory that misclassifies both positive and negative tuples. Figure 6 plots the accuracy of the domain theory and the number of literals expanded by FOCL in theory-mode, FOCL in information-mode, and FOCL with no domain theory, when the domain theory was modified by adding or deleting clauses and literals as a function of the number of modifications to the domain theory (averaged over 10 trials).

========== Figure 6 ==========

The results of adding and deleting clauses and literals indicate that FOCL in information-mode with an incorrect and incomplete domain theory generally explores a smaller portion of the search space than FOCL without a domain theory until the domain theory falls below 70% accuracy. In contrast, FOCL without information-mode requires more search than FOCL with information-mode when the domain theory is accurate. However, when the domain theory becomes extremely inaccurate, FOCL without information-mode requires less search than FOCL with information-mode and only slightly more search than FOCL with no domain theory. This occurs because there is small overhead to checking the information gain of operationalizing the domain theory, but a major benefit when the operationalization of the domain theory has greater information gain than any

Table 11: Predicate Densities

Predicate	Density
equal_rank	.125
not equal_rank	.875
adjacent_rank	.25
not adjacent_rank	.75
between_rank	.32
not between_rank	.68

inductively learned literals.

Now that we have given experimental support for the utility of various semantic constraints, we also give experimental support for the estimates of the tuple size, as developed in Section 3. To verify our estimate, we consider the task of learning the *illegal* predicate. We trace the tuple size as each literal is selected and compare it with the tuple size predicated by equation 4. This comparison is given in Table 4.[9]

As before, we can compute the density of the various background predicates which are given in Table 11.

An identical table of densities exists for the predicates involving files.

Using Table 4, we can compare the predicted values for the tuple growth. These results are from one entire episode of FOCL learning *illegal* using the background predicates in Table 11. In Table 12, we give a trace of FOCL learning *illegal* and compare the actual tuple size with the predicted tuple size.

As Table 12 indicates, the estimates are reasonable. This analysis assumed no interaction between the predicates and no intelligent choice of predicates. In the ranks marked †, the achieved tuple size is much less than the predicated one.

[9]The definition learned for illegal is:

```
illegal(A,B,C,D,E,F) :- equal_rank(E, C),not(equal_rank(C, A)).
illegal(A,B,C,D,E,F) :- equal_file(F, D),not(equal_file(D, B)).
illegal(A,B,C,D,E,F) :- adjacent_file(F, B),adjacent_rank(E, A).
illegal(A,B,C,D,E,F) :- equal_rank(E, A),adjacent_file(F, B).
illegal(A,B,C,D,E,F) :- equal_file(F, B),adjacent_rank(E, A).
illegal(A,B,C,D,E,F) :- equal_file(D, B),equal_rank(C, A).
illegal(A,B,C,D,E,F) :- equal_rank(E, A),equal_file(F, B).
illegal(A,B,C,D,E,F) :- equal_file(F, D),not(between_rank(E, A, C)).
illegal(A,B,C,D,E,F) :- equal_rank(E, C),not(between_file(F, B, D)).
```

Table 12: Predicted versus Actual Tuple Size

initial	predicate	predicted	actual
641	equal_rank	80	98
98	not equal_rank	86	87
554	equal_file	69	75
75	not equal_file	66	67
487	adjacent_file	122	102
102	adjacent_rank	25.5	24
463	equal_rank	58	70
70	adjacent_file	17.5	16
447	equal_file	56	71
71	adjacent_rank	18	13
434	equal_file	53	56
56	equal_rank	7	8
426	equal_rank	53	54
54	equal_file	7	8
†418	equal_file	52	6
6	not between_rank	4	5
†413	equal_rank	52	7
7	not between_file	5	5

In these two cases, the sample is not a mixture of positive and negative cases, but nearly entirely negative. In these two case, the bias of FOCL is to select the predicate that picks out the few positive instances.

6 Comparison to Related Work

In this section, we compare FOCL to a variety of related work on either learning relational concepts or combining empirical and inductive learning methods, focusing on the types of knowledge exploited by the systems to constrain learning and how this knowledge is used.

6.1 IOU

IOU (Mooney & Ourston, 1989) is a system that is designed to learn from overly general domain theories. IOU operates by first forming a definition via a process similar to m-EBG (Flann & Dietterich, 1989) for the positive examples. Next, IOU removes any negative examples from the training set that are correctly classified by the results of m-EBG. Finally, IOU deletes those features that are not used in the result of m-EBG from the remaining negative and all positive examples, and runs an induction algorithm on the features. The final concept is formed by conjoining the result of induction over the unexplained features with the result of m-EBG. Due to the limitations of its induction algorithm, IOU is limited to training examples expressed as attribute-value pairs as opposed to the more general relational descriptions typically used by EBL algorithms. As already mentioned, FOCL allows Horn clause descriptions of the background knowledge. In addition, the provided target concept need not be correct or overly general.

6.2 A-EBL

The A-EBL system (Cohen, 1990) is also designed to handle overly general domain theories. It operates by finding all proofs of all positive examples, and uses a greedy set covering algorithm to find a set of operational definitions that cover all positive examples and no negative examples. Unlike IOU, A-EBL will not specialize the result of m-EBG, unless required, to avoid covering any negative examples.

A similar set covering behavior occurs in FOCL when dealing with overly general domain theories caused by having superfluous clauses (see Figure 5). However, FOCL is not required to find every proof of every positive example. Furthermore, due to its induction component, FOCL can learn from overly specific domain theories as well as overly general theories caused by a clause lacking a precondition (i.e., a missing literal), in addition to overly general domain theories caused by extra clauses.

6.3 EITHER

Like FOCL, the EITHER system (Ourston & Mooney, 1990) is one of the few systems designed to work with either overly general or overly specific domain theories. Furthermore, unlike FOCL, EITHER revises incorrect domain theories, rather than just learning in spite of incorrect domain theories. EITHER contains specific operators for generalizing a domain theory by removing literals from clauses, and

by adding new clauses and operators for specializing a domain theory by adding literals to a clause. Due to its induction component and the algorithm EITHER uses to assign blame for proving a negative example or failing to prove a positive example, EITHER is restricted to using propositional domain theories and training examples represented as attribute-value pairs.

6.4 ML-SMART

In many respects, FOCL is similar to ML-SMART (Bergadano & Giordana, 1988). ML-SMART also is designed to deal with both overly general and overly specific domain theories. The major differences between ML-SMART and FOCL are involved with the search control strategy. FOCL uses hill climbing while ML-SMART uses best-first search. The best-first search may allow ML-SMART to solve some problems that cannot be solved with hill climbing, at the cost of retaining all previous states. However the cost of running a best-first algorithm is very high, being proportional to $depth^{Branching\ Factor}$. As we have already indicated by our analysis in Section 3, the branching factor grows exponentially in the length of the clauses. This means that ML-SMART will run in doubly exponential time and, therefore, is restricted to relatively small problems.

ML-SMART has a number of statistical, domain independent, and domain dependent heuristics for selecting whether to extend a rule using inductive or deductive methods. In contrast, FOCL applies a uniform information-gain metric to extensions. The heuristics in ML-SMART have not been subject to systematic experimentation of the type we performed in Section 5.5. As a consequence, it is unclear how well they deal with various types of incomplete and incorrect domain theories.

Finally, ML-SMART is only able to use its domain knowledge for explanation-based learning. In contrast, FOCL can either use domain knowledge in inductive learning, by searching for non-operational predicate variablizations. As a consequence, ML-SMART cannot learn target concepts.

6.5 FOIL

The goal of this research has been to measure the effects of adding various types of knowledge to FOIL, rather than to produce a system that performs better than FOIL. Nonetheless, direct comparison of FOIL and FOCL is possible.

Although very similar, FOCL has a slightly different control strategy from FOIL. In particular, FOCL attempts to constrain search by using variable typing, exploiting inter-argument constraints, and uses an iterative-widening approach to adding new variables. FOIL contains an admissible pruning heuristic that conflicts with the iterative-widening approach. Using variable typing, inter-argument constraints, and iterative-widening, FOCL learned the *illegal* concept by testing 1296 literals. With a domain theory, this number is reduced to 72 literals. Using the same number of examples and its pruning heuristic, FOIL requires considering 5166 literals to find a similar definition.

The typing constraints of FOCL have proved useful in improving the accuracy of the resulting hypotheses. Since these do not conflict with the pruning heuristic, they can easily be incorporated by FOIL to also reduce the search space.

The stopping criteria used by FOIL to learn from noisy data, may also be useful in stopping the learning process when there are a large number of irrelevant predicates and a small number of examples. For example, Figure 2 shows that adding irrelevant predicates decreases the accuracy of FOCL. We ran a version of FOIL provided by Quinlan on the same 60, 100, 200, and 641 training examples, learning *illegal* both with and without 20 irrelevant predicates. The accuracy of the resulting hypotheses were 92.9, 96.4, 97.4 and 99.3 with only relevant predicates and 86.1, 90.2, 96.1, 99.0 with irrelevant predicates added. With fewer than 200 examples, FOIL typically underfit a concept, yielding a rule that was not consistent with all the training examples. This is due to the stopping criterion that partially mitigates the effects of introducing irrelevant predicates into the concept description. However, the accuracy of the results of FOIL does decrease when operational predicates are introduced.

7 Conclusions

In this paper we have described a concept learner, FOCL, that combines inductive and analytic learning in a uniform manner. The resulting program employs a number of different types of knowledge. In particular, it advantageously uses both inconsistent and incomplete theories. We provided both a mathematical and an experimental evaluation of FOCL.

From our mathematical analysis, we can draw a number of important conclusions about the complexity of learning rules and the value of different sorts of knowledge. Some of these conclusions are summarized here:

- The branching factor grows exponentially in the arity of the predicate to be learned.
- The branching factor grows exponentially in the arity of the available predicates.
- The branching factor grows exponentially in the number of new variables introduced.
- The number of available predicates increases the branching factor by a linear amount.
- The difficulty in learning a rule is primarily determined by the difficulty in learning the longest Horn clause, where length is measured in the number of new variables.

- The difficulty in learning a rule is only linearly proportional to the number of clauses in the rule.
- Partial operational rules that do not include the longest clause barely reduce the search in finding the rule.
- Typing knowledge provides an exponential decrease in the amount of search necessary to find a rule.

In addition to supporting the theoretical claims made above, our experimental evidence suggests a number of other important conclusions.

- Non-operational predicates aid by improving the shape of the hill-climbing landscape.
- Any method (argument constraints, semantic constraints, typing, symmetry, etc.) that eliminates fruitless paths will decrease the search cost and increase the accuracy.
- The uniform evaluation function applied to literals learned by induction or by explanation-based methods allows FOCL to tolerate domain theories that are both incorrect and incomplete.
- Irrelevant background predicates marginally slow learning and marginally decrease accuracy, since the system has more opportunities to make incorrect decisions. In this respect, irrelevant predicates in Horn clause learning are similar to irrelevant attributes in propositional learning.
- Iterative widening reduces the cost of search while maintaining the accuracy of the resulting rule.
- A domain theory that consists of rules that are overly general by virtue of having superfluous clauses is the easiest to tolerate. In this case, only a subset of the operationalizations are needed and the information-gain metric of FOCL selects, in a greedy manner, operationalizations that cover positive examples. Other forms of incomplete and incorrect domain theories require FOCL to use induction to overcome domain theory errors.

Acknowledgements

This research is partially supported by NSF grant IRI-8908260. We would like to thank Ross Quinlan for his advice on FOIL, Dan Hirschberg for deriving the *new* recurrence, and Cliff Brunk, Tim Cain, Caroline Ehrlich, Ross Quinlan, Wendy Sarrett and Glenn Silverstein for reviewing a draft of this paper.

References

Bergadano, F., & Giordana, A. (1988). A knowledge intensive approach to concept induction. *Proceedings of the Fifth International Conference on Machine Learning* (pp. 305–317). Ann Arbor, MI: Morgan Kaufmann.

Bergadano, F., Giordana, A., & Ponsero, S. (1989). Deduction in top-down inductive learning. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 23–25). Ithaca, NY: Morgan Kaufmann.

Cohen, W. (1990). *Abductive explanation-based learning: A solution to the multiple explanation-problem* (ML-TR-29). New Brunswick, NJ: Rutgers University.

Danyluk, A. (1989). Finding new rules for incomplete theories: Explicit biases for induction with contextual information. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 34–36). Ithaca, NY: Morgan Kaufmann.

Fisher, D. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning, 2*, 139–172.

Flann, N., & Dietterich, T. (1989). A study of explanation-based methods for inductive learning. *Machine Learning, 4*, 187–226.

Ginsberg, M., & Harvey, W. (1990). Iterative broadening. *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 216–220). Boston, MA: Morgan Kaufmann.

Hirsh, H. (1989). Combining empirical and analytical learning with version spaces. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 29–33). Ithaca, NY: Morgan Kaufmann.

Katz, B. (1989). Integrating learning in a neural network. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 69–71). Ithaca, NY: Morgan Kaufmann.

Korf, R. E. (1985). Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence, 1*, 11–46.

Lebowitz, M. (1986). Integrated learning: Controlling explanation. *Cognitive Science, 10.*

Michalski, R. (1980). Pattern recognition as rule-guided inductive inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 2*, 349–361.

Mitchell, T., Keller, R., & Kedar-Cabelli, S. (1986). Explanation-based learning: A unifying view. *Machine Learning, 1*, 47–80.

Mooney, R., & Ourston, D. (1989). Induction over the unexplained: Integrated learning of concepts with both explainable and conventional aspects. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 5–7). Ithaca, NY: Morgan Kaufmann.

Muggleton, S., Bain, M., Hayes-Michie, J., & Michie, D. (1989). An experimental comparison of human and machine learning formalisms. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 115–118). Ithaca, NY: Morgan Kaufmann.

Muggleton, S., & Buntine, W. (1988). Machine invention of first-order predicates by inverting resolution. *Proceedings of the Fifth International Workshop on Machine Learning* (pp. 339–352). Ann Arbor, MI: Morgan Kaufmann.

Ourston, D., & Mooney, R. (1990). Chaining the rules: A comprehensive approach to theory refinement. *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 815–820). Boston, MA: Morgan Kaufmann.

Pagallo, G., & Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning, 5*, 71–100.

Pazzani, M. (1989). Explanation-based learning with weak domain theories. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 72–74). Ithaca, NY: Morgan Kaufmann.

Pazzani, M. J. (1990). *Creating a memory of causal relationships: An integration of empirical and explanation-based learning methods.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Pazzani, M., & Brunk, C. (1990) Detecting and correcting errors in rule-based expert systems: An integration of empirical and explanation-based learning. *Proceedings of the Workshop on Knowledge Acquisition for Knowledge-Based System.* Banff, Canada.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1*, 81–106.

Quinlan, J. R. (1989). *Learning relations: Comparison of a symbolic and a connectionist approach* (Technical Report). Sydney, Australia: University of Sidney.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning, 5*, 239–266.

Sarrett, W., & Pazzani, M. (1989). One-sided algorithms for integrating empirical and explanation-based learning. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 26–28). Ithaca, NY: Morgan Kaufmann.

Shavlik, J., & Towell, G. (1989). Combining explanation-based learning and artificial neural networks. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 90–93). Ithaca, NY: Morgan Kaufmann.

Widmer, G. (1990). *Incremental knowledge-intensive learning: A case study based on an extension to Bergadano & Giordana's integrated learning strategy* (Technical Report). Austrian Research Institute for Artificial Intelligence.

Appendix I A trace of FOCL

```
king_attack_king(R1,F1,R3,F3)                          +  61.6
rook_attack_king(R1,F1,R2,F2,R3,F3)                    + 124.1
same_loc(R1,F1,R2,F2),adjacent_file(F1,F3)             +   0.0
same_loc(R2,F2,R3,F3)                                  +  18.9
same_loc(R1,F1,R3,F3)                                  +  11.6

OPERATIONALIZING rook_attack_king(R1,F1,R2,F2,R3,F3)
OPERATIONALIZING equal_file(F2,F3),king_not_between_rank(R1,F1,R2,F2,F3)
OPERATIONALIZING king_not_between_rank(R1,F1,R2,F2,F3)

equal_file(F1,F2),not_between_rank(R2,R1,R3)           +   0.135
not(equal_file(F1,F2))                                 +   1.299

CLAUSE 1 illegal(R1,F1,R2,F2,R3,F3):-equal_file(F2,F3),not(equal_file(F1,F2))
-------------------------------------------------------------------------------

king_attack_king(R1,F1,R3,F3)                          +  70.7
rook_attack_king(R1,F1,R2,F2,R3,F3)                    +  14.8
same_loc(R1,F1,R2,F2),adjacent_file(F1,F3)             +   0.0
same_loc(R2,F2,R3,F3)                                  +   1.8
same_loc(R1,F1,R3,F3)                                  +  14.8

OPERATIONALIZING king_attack_king(R1,F1,R3,F3)
CLAUSE 2 illegal(R1,F1,R2,F2,R3,F3):-adjacent_rank(R1,R3),adjacent_file(F1,F3)
-------------------------------------------------------------------------------

CLAUSE 3 illegal(R1,F1,R2,F2,R3,F3):-adjacent_rank(R1,R3),equal_file(F1,F3)
-------------------------------------------------------------------------------

king_attack_king(R1,F1,R3,F3)                          +  23.9
rook_attack_king(R1,F1,R2,F2,R3,F3)                    +  13.1
same_loc(R1,F1,R2,F2),adjacent_file(F1,F3)             +   0.0
same_loc(R2,F2,R3,F3)                                  +   0.0
same_loc(R1,F1,R3,F3)                                  +  17.5

OPERATIONALIZING king_attack_king(R1,F1,R3,F3)
adjacent_rank(R1,R3),adjacent_file(F1,F3)              +   0.0
equal_rank(R1,R3)                                      +  26.7
knight(R1,F1,R3,F3)                                    +  -0.1
adjacent_rank(R1,R3),equal_file(F1,F3)                 +   0.0
BEST CONDITION equal_rank(R1,R3)

between_file(F1,F2,F1)                                 +   0.0;  -   0.0
between_file(F1,F2,F3)                                 +  -3.26; -   4.6
...
equal_file(F3,F1)                                      +  10.17; -  _6.1
adjacent_file(F3,F1)                                   +  20.34; - -10.2

CLAUSE 4 illegal(R1,F1,R2,F2,R3,F3):-equal_rank(R1,R3),adjacent_file(F3,F1)
-------------------------------------------------------------------------------

CLAUSE 5 illegal(R1,F1,R2,F2,R3,F3):-equal_rank(R1,R3),equal_file(F1,F3)
-------------------------------------------------------------------------------
CLAUSE 6 illegal(R1,F1,R2,F2,R3,F3):-equal_file(F3,F2),equal_file(F1,F2)
                                     not(between_rank(R2,R1,R3))
-------------------------------------------------------------------------------
```

```
king_attack_king(R1,F1,R3,F3)                          +  -1.0
rook_attack_king(R1,F1,R2,F2,R3,F3)                    +   0.0
same_loc(R1,F1,R2,F2),adjacent_file(F1,F3)             +   0.0
same_loc(R2,F2,R3,F3)                                  +   0.0
same_loc(R1,F1,R3,F3)                                  +   0.0

between_file(F1,F2,F3)                                 +   5.5;  -  -5.0
...
equal_rank(R3,R2)                                      + 192.4;  - -25.3
BEST CONDITION equal_rank(R3,R2)

king_attack_king(R1,F1,R3,F3)                          +  -2.1
...
between_file(F1,F2,F3)                                 +   0.9;  -  -0.9
...
equal_rank(R1,R2)                                      +  -2.2;  -   2.6


CLAUSE 7 illegal(R1,F1,R2,F2,R3,F3):- equal_rank(R3,R2),not(equal_rank(R1,R2))
-------------------------------------------------------------------------------
king_attack_king(R1,F1,R3,F3)                          +   4.5
rook_attack_king(R1,F1,R2,F2,R3,F3)                    +   0.0
same_loc(R1,F1,R2,F2),adjacent_file(F1,F3)             +   0.0
same_loc(R2,F2,R3,F3)                                  +   0.0
same_loc(R1,F1,R3,F3)                                  +   0.0

OPERATIONALIZING king_attack_king(R1,F1,R3,F3)

adjacent_rank(R1,R3),adjacent_file(F1,F3)              +   0.0
equal_rank(R1,R3)                                      +   9.0
knight(R1,F1,R3,F3)                                    +   0.0
adjacent_rank(R1,R3),equal_file(F1,F3)                 +   0.0

BEST CONDITION equal_rank(R1,R3)

between_file(F1,F2,F3)                                 +   4.4;  - -1.6
equal_rank(R3,R2)                                      +  13.5;  -  0.0
...
BEST condition equal_rank(R3,R2)

between_file(F1,F2,F3)                                 +   1.9;  - -1.1
between_file(F2,F1,F3)                                 +   0.0;  -  2.4


CLAUSE 8 illegal(R1,F1,R2,F2,R3,F3):-equal_rank(R1,R3), equal_rank(R3,R2),
                                     not(between_file(F2,F1,F3)).

-------------------------------------------------------------------------------

CLAUSE 9 illegal(R1,F1,R2,F2,R3,F3):-equal_file(F2,F1),equal_rank(R2,R1)
```

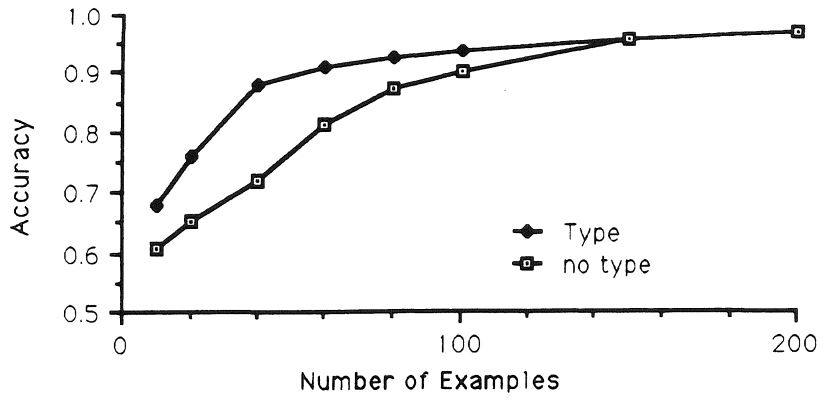

Figure 1. The effect of variable typing on the accuracy of the FOCL.

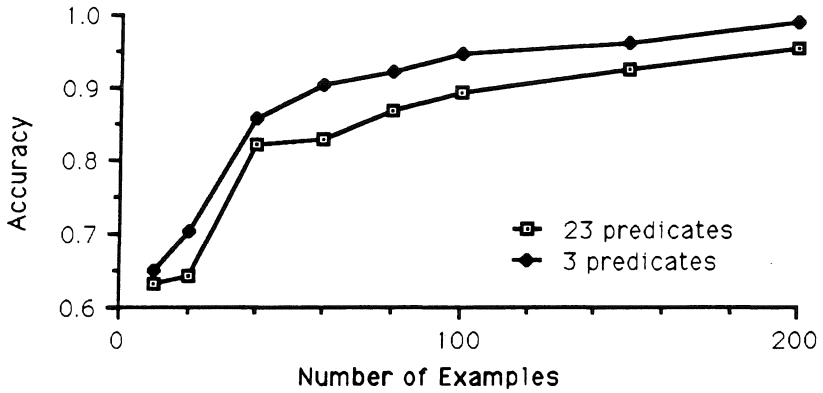

Figure 2. The effect of adding irrelevant predicates on the accuracy of the FOCL.

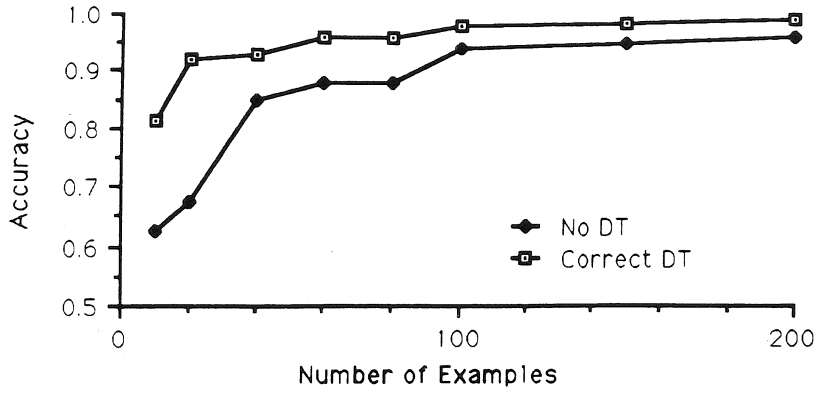

Figure 3. The effect of adding a correct domain theory and target concept on the accuracy of the FOCL.

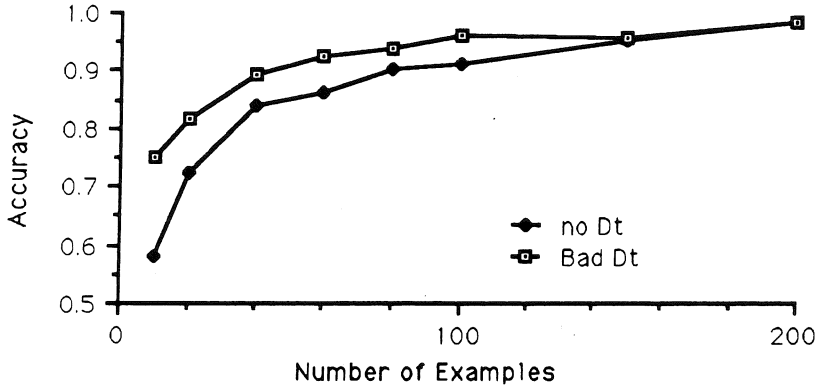

Figure 4. The effect of adding a domain theory that is 76.2% accurate on the accuracy of the FOCL.

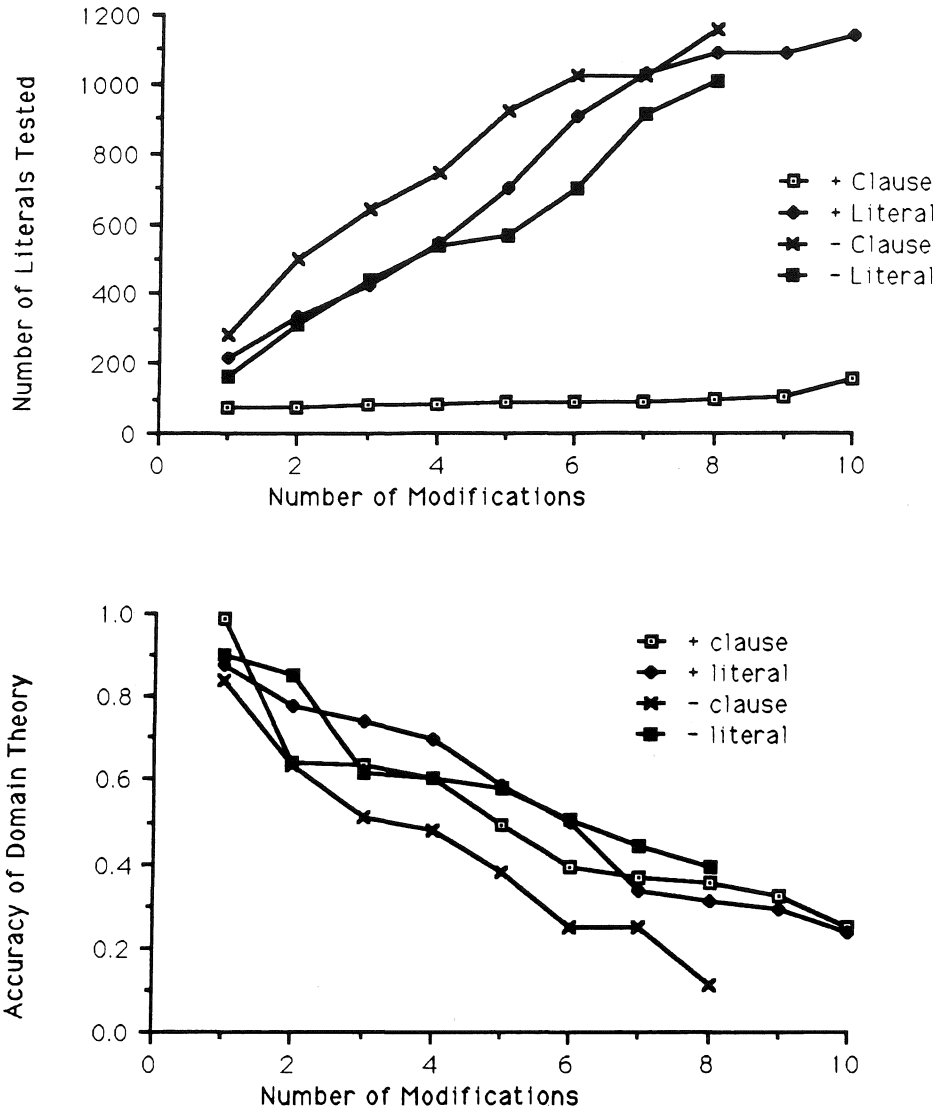

Figure 5. Upper: The effect of modifying a domain theory by individually adding literals, deleting literals, adding clauses and deleting clauses on the amount of search required by FOCL in theory-mode to learn an accurate concept. Lower: The accuracy of the modified domain theory.

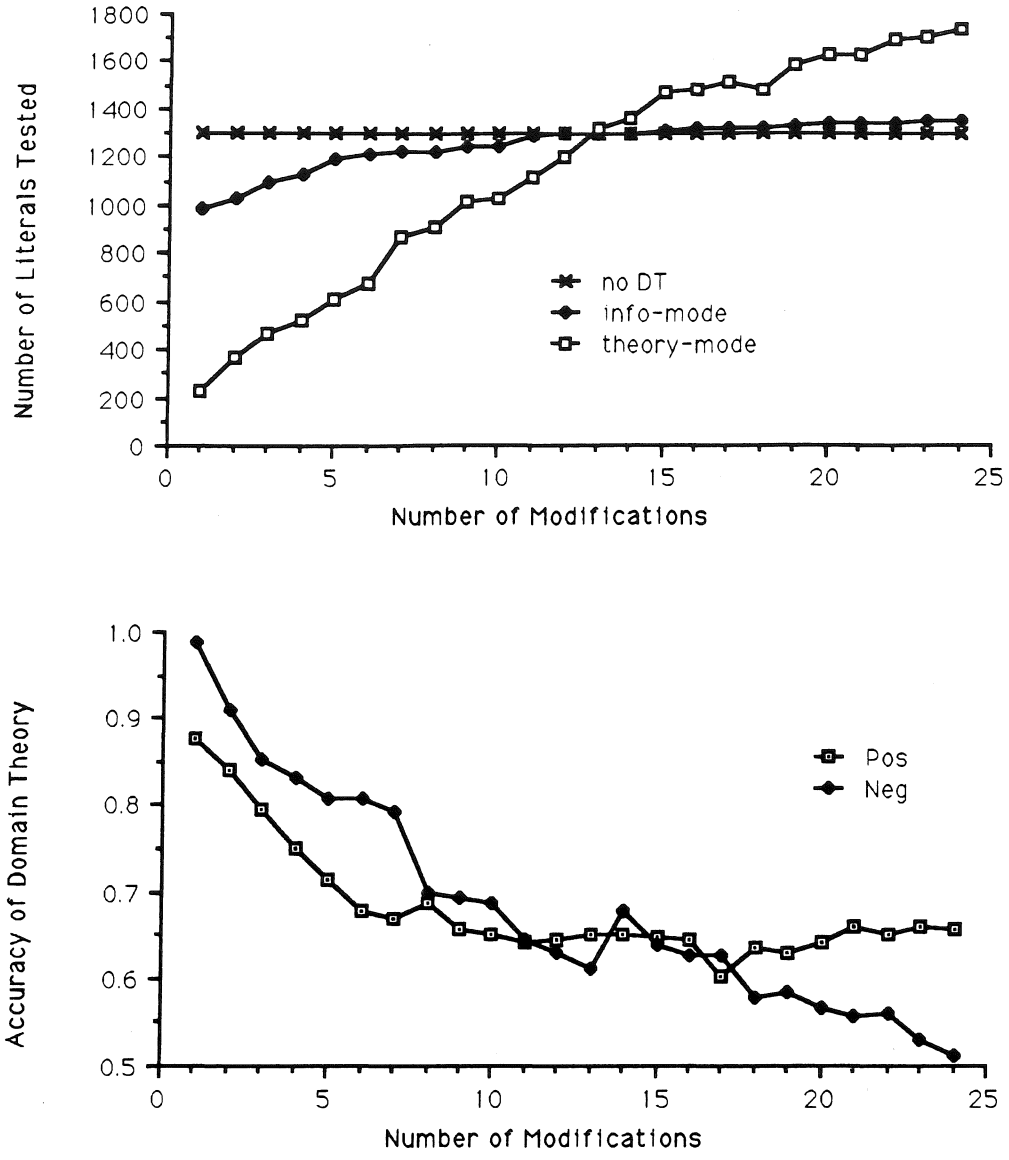

Figure 6. The effect of modifying a domain theory by combinations of adding literals, deleting literals, adding clauses and deleting clauses on the accuracy of the domain theory and on the amount of search required by FOCL with no domain theory, FOCL in theory-mode and FOCL in information-mode to learn an accurate concept.