Machine Learning, 9, 57–94 (1992)

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 92717-3425

Editor: Paul Rosenbloom

Abstract. In this paper, we demonstrate how different forms of background knowledge can be integrated with an inductive method for generating function-free Horn clause rules. Furthermore, we evaluate, both theoretically and empirically, the effect that these forms of knowledge have on the cost and accuracy of learning. Lastly, 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.

Keywords: Learning relations, combining inductive and explanation-based learning

1. Introduction

There are two general approaches to concept learning studied in artificial intelligence. Empirical learning programs operate by finding regularities among a group of training examples. One weakness of this approach is that in its purest form, it cannot take advantage of the existing knowledge of the learner. Explanation-based learning (EBL)systems (De Jong & Mooney, 1986; Mitchell, Kellar, & Kedar-Cabelli, 1986) operate by using a domain theory[1] to explain a single example, and forming a general description of the class of examples with the same explanation. One weakness of EBL is that the learned concept description cannot be more accurate than the domain theory. Integrated learning systems, (i.e., systems that combine empirical and explanation-based learning) have the potential of overcoming the weakness of either method applied individually.

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 the 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 definition 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 function-free Horn clauses, a useful subset of first-order predicate calculus. In Section 2,

we review FOIL and, in the next section, we analyze the complexity of FOIL in terms of the size of the 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.

In Section 3, we incrementally define an extension to FOIL which combines both inductive and explanation-based learning. The new learning system called FOCL (First Order Combined Learner) 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, with prior knowledge, learns more accurate Horn clause concept definitions and with less computational costs.

Given a set of examples and a correct domain theory, the output of FOCL is similar to that of applying explanation-based learning on each unexplained positive example. Conversely, without any background knowledge, FOCL operates like FOIL. In Section 5, we describe how FOCL learns with 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 while others are added analytically).

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. In Section 5, 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.

Throughout this paper we use the simple domain of list relations to illustrate FOCL's learning mechanisms. A more severe test is provided by the more complex domain of chess end games. This was the most difficult problem on which FOIL has been tested. A domain theory for this particular chess problem is succinct and amenable to systematic experimentation by mutation. 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).

Section 6 relates FOCL to other systems that combine empirical and explanation-based learning. Section 7 summarizes FOCL's current characteristics and suggests future extensions.

2. Background: FOIL

In this section, we review FOIL so that we can analyze its complexity. We begin by introducing some definitions. Formally, 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. FOIL permits predicates with variables but does not allow them to contain function symbols, including constants. Syntactically, a *literal* is a predicate or its negation. Semantically, FOIL adopts PROLOG's negation-as-failure rule to define the meaning of a negated predicate. Under this interpretation, if no proof that a tuple satisfies

the predicate exists, then the tuple satisfies the negated predicate. Literals that are unnegated predicates are called *positive literals*. Literals that are the 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 $P \leftarrow L_1, L_2, \ldots$ 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 a 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 do 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,[2] but does allow negated predicates. FOIL also permits restricted use in clause bodies of the predicate it is learning. This allows FOIL to 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. Separate-and-conquer approaches concentrate on creating one rule at a time, collecting the uncovered examples into a single pot that will be handed to the induction algorithm again.

Table 1 presents a high-level view of the FOIL algorithm. The algorithm has two main stages: separate and conquer. 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.

To more precisely present FOIL's algorithm, we need to define carefully 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 (Body in Table 1) is *grandfather*$(X, Y) \leftarrow$ *parent*(X, Z). This clause can be extended 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 be selected, but also a particular set of variables for the predicate-name. We call the choice of variables for a predicate-name a *variabilization* 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),

Table 1. FOIL Design I.

```
Let Pred be the predicate to be learned.
Let Pos be the positive examples.
Until Pos is empty do:
  Let Neg be the negative examples.
  Set Body to empty.
  Call LearnClauseBody.
  Add Pred ← Body to the rule.
  Remove from Pos all examples that satisfy the Body.

Procedure LearnClauseBody
  Until Neg is empty do:
    Choose a literal L.
    Conjoin L to Body.
    Remove from Neg examples that do not satisfy L.
```

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.

For example, 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 fact 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 variabilization 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 variabilizations of every extensionally defined predicate.[3]

3. Analysis of FOIL

In general, the cost of 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,

Table 2. FOIL Design II.

```
Let Pred be predicate to be learned.
Let Pos be the positive tuples.
Until Pos is empty do:
  Let Neg be the negative tuples.
  Let Body be empty.
  Let Old be those variables used in Pred.
  Call LearnClauseBody.
  Add Pred ← Body to the rule.
  Remove from Pos all tuples that satisfy the Body.

Procedure LearnClauseBody
  Until Neg is empty do:
    For each predicate-name P.
    For each variabilization L of P.
    Compute information gain of L and its negation.
    Select literal L with most information gain.
    Conjoin L with Body.
    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.
```

while not constant, is at least bounded. In FOIL, 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 of FOIL

In order to compute the number of different literals to be considered for evaluation, let us first consider the number of different variabilizations of a single predicate P of arity m when the current clause has k old variables. Let this number be $v(m, k)$. In Appendix I we provide a detailed analysis of this value. At this point we present an approximation which will be sufficient for a qualitative understanding. The first few values of $v(m, k)$ are displayed in Table 3, indicating the rapid growth of $v(m, k)$ in both m and k.

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

$v(m, k)$	OLD variables							
Arity	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	10,375	20,826	38,647	$\approx k^5$
6	674	3060	10,278	29,168	72,810	163,764	338,030	$\approx k^6$

Let $Pred(i)$ be the number of predicates of arity i. Let $MaxA$ 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:

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

The factor of 2 reflects the fact that for each predicate we also consider its negation.

Although $v(m, k)$ is complex (see Appendix I), we can easily compute an upper bound. Let *Old* be the maximum number of old variables. Let *AllPred* be the total number of predicates. To add a new predicate we may choose from one of *AllPred* predicates. If the predicate has arity *MaxA* (the worst case), then we must consider choosing *MaxA* variables from *Old* old variables and $MaxA - 1$ new variables. A simple upper bound on this cost is: $(Old + MaxA - 1)^{MaxA}$. Consequently, an upper bound of the theory cost is:

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

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 illustrate this analysis by measuring the complexity of learning a particular concept, for example, the concept of list membership. This domain, as defined by Quinlan (1990), has three predicates, *null*, *member*, and *component*, with arities one, two, and three, respectively. Components (A, B, C) is true if, in PROLOG notion $[A \mid B] = C$. The definition of member is given in Table 4.

Let us concentrate our attention on learning the boxed conjunct of the second clause, which is the point at which the branching factor is maximized. At this point the number of old variables is four. There are three available predicates, one of each arity from one to three. Using Table 3 we see that the number of extensions is $2 * (1 * 4 + 1 * 24 + 1 * 136) = 328$.

Table 4. Definition of member.

member(*X*, *Y*) ← *component*(*X*, *Z*, *Y*).
member(*X*, *Y*) ← *component*(*A*, *B*, *Y*), member(X, B) .

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 *MaxP* be the number of predicates with largest arity *MaxA*. Then an approximation of the total number of nodes generated to learn *R* is:

$$NodesSearched \approx 2 * MaxP * v(MaxA, Var) \leq 2 * MaxP * (Var + MaxA - 1)^{MaxA}.$$

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

3.2. Evaluation cost of FOIL

In the previous section, we computed the number of different extensions of the current clause. Each 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. Regardless how it is implemented, we suppose that this cost is a proportional to 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 only one 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 new tuple size, we 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 one to ten. Then the density of the *successor*(*X*, *Y*) predicate is 9/100 and the density of *less*(*X*, *Y*) is 45/100. If a literal introduces no new variables, then the tuple size will not increase as it consists of the subset of the current tuples that 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 when 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 one to ten, the power is nine, which is achieved by *less*(1, *Y*) and by *less*(*X*, 10). Similarly, the power of the *successor* predicate is one. 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. Consequently, we expect that the tuple size grows proportional to AveragePower, rather than Power.

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

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

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 Section 5.1, Tables 11 and 12 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. Indeed, to bound the induction process, GOLEM (Muggleton & Feng, 1990) restricts its predicates in exactly this manner.

In the subsequent sections, we will show 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: Adding knowledge to FOIL

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 use domain knowledge to guide the learning process. One set of extensions allows FOCL to use constraints to limit the search space. A second set 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 intensionally defined predicates is identical to the domain theory of EBL. 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. Unlike FOIL, FOCL integrates background knowledge and EBL methods with an inductive learner.

In the following sections, we describe these extensions in more detail and evaluate the effect of each extension on either the number of literals tested by FOCL or the accuracy of FOCL. To illustrate these extensions, we use two domains. In the first domain, that of list relations, we illustrate how FOCL learns a simple recursive concept, the *member* predicate. FOCL 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 more complicated and was introduced by Muggleton and Feng (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*). This is true if a chess board containing a white king and rook and black king is in an illegal state if it is white's turn to move. 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 grow 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 this fails because no variabilization of any predicate has positive gain, then additional free variables are allowed. On each failure, an additional free variable is allowed until the number of free variables exceeds the maximum arity of any predicate. As with iterative deepening, 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 variabilizations of some predicates. 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. Moreover, it can 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.[5] 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. At this point, we present the specification of FOCL in Table 5.

The subsequent sections discuss the modifications necessary to use each of the forms of knowledge that may be presented to FOCL.

Table 5. 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 intentionally 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.

4.2. Single argument constraints

Type constraints provide a useful and inexpensive way of incorporating a simple form of background knowledge. FOCL can easily use typing information.[6] 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:

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, the theory cost approximately reduces to $TheoryCost = AllPred^*((Var + MaxA)/T)^{MaxA}$, a savings of T^{MaxA}. This shows that, in the best case, typing can reduce the exponent of the search space. 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.

In order to test the hypothesis that typing can improve the accuracy of FOCL, we ran an experiment in which we compared FOCL without typing to FOCL with typing on the *illegal* problem. Twenty trials of FOCL with typing and without typing were run on 10, 20, 40, 60, 80, 100, 150, 200, 250 and 300 randomly selected training examples. We measured the accuracy at each point on 1000 randomly selected testing examples. Figure 1 shows the mean accuracy plotted as a function of the number of training examples. FOCL with typing is labeled "Typed Induction." FOCL without typing is labeled "Untyped." ("Irrelevant Pred" will be used in the next experiment.) An analysis of variance indicates that typing has a significant effect on the accuracy of FOCL ($F(1,380) = 75.8$, $p = < .0001$). A similar effect could be achieved by not permitting a data value to belong to two types. In the representation used for *illegal*, numbers are used to represent the ranks and files. If numbers were used for the ranks, and letters for files, it wouldn't be possible for a rank to have the same value as a file. However, without explicit typing, FOCL would still have to consider the possibility that a rank could equal a file.

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 variabilization 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.

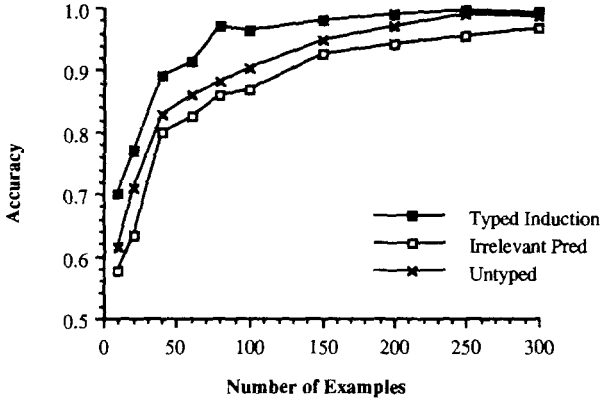

Figure 1. The effect of adding irrelevant predicates and variable typing on the accuracy of FOCL.

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 multiple argument constraint we have implemented handles predicates where all the variables must be distinct. This is declared in the input to FOCL via a *unique_variables* declaration on designated predicates. 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 variabilized 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 220 CPU seconds on a SUN 4/65 computer running Common Lisp. With these constraints, 21.6 CPU seconds were consumed during learning *illegal*. This illustrates that both the run time and the size of the search space explored are reduced by an order of magnitude by these simple constraints on inductive learning.

Another implemented inter-argument constraint is the requirement that for some predicates, variables must commute. 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 variabilizations 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 variabilizations of the same literal.

Finally, we have also implemented a third constraint that reduces the number of variabilizations of a predicate that must be checked. A predicate may have a *mode* declared. The mode indicates whether each argument to the predicate may be bound to an old variable or a new variable. Mode information is commonly used by a compiler to create more efficient compiled predicates. In FOCL, we use it to avoid computing the information gain of variabilizations of a predicate that violate the mode restriction. The mode interacts with the iterative widening search used in FOCL, because FOCL initially considers only those variabilizations that have no new variables. As a consequence, it does not decrease the amount of search performed on the *illegal* problem. However, when new variables are needed in a concept representation, and modes are declared for predicates, the mode declarations can significantly decrease the amount of search performed during learning.

4.4. Operational initial rules

In the next sections, we consider ways in which background knowledge can improve 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. If any clause has positive information gain, the conjunction of operational literals is added to the end of the clause under construction. If the current clause covers some negative tuples, additional literals are added inductively 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. For the first definition, FOCL tests 268 literals and considers 20,140 tuples. In the second definition, the corresponding figures are 228 literals and 12,167 tuples. Using the third definition, these figures are 311 literals and 23,358 tuples. Without any partial definition, FOIL tests 308 literals and considers 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 only slightly increases the number of literals and tuples tested since FOCL checks the possibility that the partial definition may have some information gain, and ignores the definition when there is not positive gain.

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. Note that an initial partial operational concept definition might improve the accuracy of FOIL if FOIL could not find an accurate concept definition using greedy search.

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. Non-operational predicates will bias the search for a concept definition, but do not appear in the final concept definition. 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*) ← *less_than*(*X, Y*), *less_than*(*Y, Z*).
between(*X, Y, Z*) ← *less_than*(*Z, Y*), *less_than*(*Y, X*).

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 variabilizations 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 variabilization includes new variables) covered by the literal.[7] 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 potentially expensive Prolog proof.

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 construct a proof tree which is likely to cover many cases. This process is formalized in Table 6.

Table 6. Operationalization.

```
Procedure: Operationalize(Literal, Pos, Neg)
If Literal is operational
  Return Literal
Initialize OperationalLiterals to the empty set.
For each clause in the definition of Literal
  compute_gain(clause, Pos, Neg).
For the clause with the maximum gain,
  for each literal L in the clause,
    Add operationalize(L, Pos, Neg) to OperationalLiterals.
```

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 a 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 7 contains definitions of these non-operational predicates.

With this extended domain theory, FOCL tested 3063 literals and 283,602 tuples before finding an operational concept definition, as opposed to 1296 literals and 109,350 tuples when only the operational predicates were used. This experimental finding agrees with the analysis of FOIL presented in Section 3. In particular, since the number of predicates increased, the number of literals tested increased.

A slight modification to the operationalization procedure described so far increases FOCL's ability to tolerate overly specific domain theories caused by clauses having one or more extra literals. In particular, the information gain of the conjunction of literals produced by operationalization may be increased by the deletion of one of the literals of the conjunction. When deleting a literal increases the information gain and the ratio of negative tuples to total tuples is decreased by the deletion, then the literal is deleted from the operationalization. This process is repeated until no deletion results in additional information gain. Note that this scheme is a greedy search for the subset of an operationalization with the maximum information gain. An optimal algorithm that is guaranteed to find the subset with the maximum information gain would operate by finding the information gain of all subsets of the operationalization. However, this expensive scheme is not practical in large applications. In Pazzani, Brunk, and Silverstein (1991), we provide experimental evidence that the greedy method is an efficient approximation of the optimal algorithm.

Table 7. 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*) ← *equal_rank*(*R1,R2*), *equal_file*(*F1,F2*).
type: (*rank file rank file*), *unique variables.*
king_attack_king(*R1,F1,R2,F2*) ← *adjacent_rank*(*R1,R2*), *adjacent_file*(*F1,F2*)
king_attack_king(*R1,F1,R2,F2*) ← *adjacent_rank*(*R1,R2*), *equal_file*(*F1,F2*)
king_attack_king(*R1,F1,R2,F2*) ← *equal_rank*(*R1,R2*), *adjacent_file*(*F1,F2*)
type: (*rank file rank file*), *unique_variables.*
rook_attack_king(*R1,F1,R2,F2,R3,F3*) ← *equal_rank*(*R2,R3*), *king_not_between_file*(*R1,F1,R2,F2,F3*).
rook_attack_king(*R1,F1,R2,F2,R3,F3*) ← *equal_file*(*F2,F3*), *king_not_between_rank*(*R1,F1,R2,F2,F3*).
type: (*rank file rank file rank file*), *unique_variables.*
king_not_between_file(*R1,F1,R2,F2,F3*) ← *not*(*equal_rank*(*R1,R2*)).
king_not_between_file(*R1,F1,R2,F2,F3*) ← *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*) ← *not*(*equal_file*(*F1,F2*)).
king_not_between_rank(*R1,F1,R2,F2,R3*) ← *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)

A positive effect of a domain theory is that it may 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 variabilization of an irrelevant predicate may have the maximum information gain, and be used incorrectly as a literal in a clause. This problem is not limited to just non-operational predicates. This is analogous to an irrelevant attribute in propositional learning.

In order to test the hypothesis that irrelevant predicates degrade the accuracy of FOCL, we ran an experiment in which we compared FOCL without irrelevant predicates to FOCL with irrelevant predicates on the *illegal* problem. Irrelevant predicates such as *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*) were added. Both versions of FOCL did not use typing. The experiment follows the same design as the previous experiment. Figure 1 also plots the mean accuracy of FOCL with irrelevant predicates. An analysis of variance indicates that irrelevant predicates have a significant effect on the accuracy of FOCL ($F(1,380) = 37.6$, $p < .0001$). 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 are not known 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.

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 a *target concept*, an abstract description of the concept to be learned. For example, Table 8 shows a representation of an initial non-operational rule (target concept) for *illegal*. In EBL, the target concept is assumed to be correct. In FOCL, we relax the assumptions that the target concept and the domain theory are correct.

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. If a clause of the initial rule is non-operational and has maximum gain, literals are added to the current clause by operationalization of the target concept in the manner described in Section 4.5.

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 variabilization of every predicate. When provided with a target concept, the proof structure determines which variabilizations are tested. FOCL computes the information gain of *illegal*(*R*1, *F*1, *R*2, *F*2, *R*3, *F*3) and operationalizes it if it has positive information gain. As a result, only three variabilizations of *same_loc* are tested when the target concept is provided to FOCL. Without this knowledge, FOCL tests 2712 different variabilizations of *same_loc*, (since there are six variables and *same_loc* has arity four). Without the target concept, the constraints of typing and unique variables, and iterative widening search reduce this number to 36. Similarly, unconstrained induction checks 163,764 variabilizations of *rook_attack_king*, constrained induction checks 36, and operationalization checks only one.

Table 8. Chess target concept.

illegal(*R*1,*F*1,*R*2,*F*2,*R*3,*F*3) ← *same_loc*(*R*1,*F*1,*R*2,*F*2).
illegal(*R*1,*F*1,*R*2,*F*2,*R*3,*F*3) ← *same_loc*(*R*1,*F*1,*R*3,*F*3).
illegal(*R*1,*F*1,*R*2,*F*2,*R*3,*F*3) ← *same_loc*(*R*2,*F*2,*R*3,*F*3).
illegal(*R*1,*F*1,*R*2,*F*2,*R*3,*F*3) ← *king_attack_king*(*R*1,*F*1,*R*3,*F*3).
illegal(*R*1,*F*1,*R*2,*F*2,*R*3,*F*3) ← *rook_attack_king*(*R*1,*F*1,*R*2,*F*2,*R*3,*F*3).

Note that typing, unique variables, and iterative widening are not needed by the analytic learning component, since the domain theory and the target concept control the selection of predicate variabilizations. 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 improve accuracy. In order to test the hypothesis that a correct domain theory will increase the accuracy of FOCL, we ran an experiment in which we compared FOCL without a domain theory to FOCL with a domain theory for *illegal*. Although typing is always valuable, to focus just on the value of the domain theory in this experiment, both versions of FOCL did not use typing. The experiment follows the same design as the previous experiments. Figure 2 plots the mean accuracy of FOCL with and without a correct domain theory. FOCL with a correct domain theory is labeled "Correct DT." FOCL without a correct domain theory is labeled "Untyped." ("Incorrect DT" will be used in the next experiment.) An analysis of variance indicates that typing has a significant effect on the accuracy of FOCL ($F(1,380) = 75.8$, $p < .0001$). An analysis of variance indicates that a correct domain theory has a significant effect on the accuracy of FOCL ($F(1,380) = 337.9$, $p < .0001$). As expected, the correct target concept and domain theory improve the accuracy of the resulting hypothesis. FOCL does require close to 200 examples to typically achieve 100% accuracy with a correct domain theory on this problem because some of the clauses in the correct concept are rare and FOCL requires that at least one positive example be covered by any operationalization.

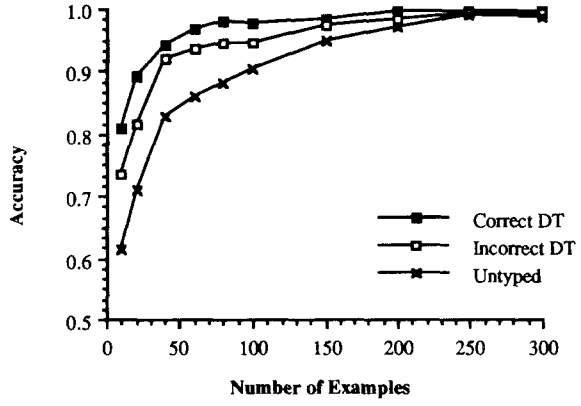

Figure 2. The effect of adding a correct domain theory and a 76.2% accurate domain theory on the accuracy of FOCL.

A domain theory and target concept does reduce the CPU time in FOCL. A total of 11.6 CPU seconds on a Sun 4/60 were consumed using purely explanation-based learning as compared to 220 CPU 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 1500 logical inferences per second.

4.7. Summary of FOCL

Now that the components of FOCL have been explained, we show how they are integrated in Table 9. This high level design emphasizes the main differences with FOIL. FOCL extends FOIL in several ways. First, FOCL constrains the inductive process so that not all variabilizations of a predicate need be checked. Second, FOCL computes the information gain of intensionally defined predicates as well as extensionally defined predicates. Third, FOCL operationalizes intensionally defined predicates by finding an operational specialization that covers many positive and few negative examples. Fourth, FOCL computes the information gain of an initial (operational or non-operational) rule for the concept to be learned and can 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 variabilizations of a non-operational predicate that are likely to be useful. For simplicity, we do not consider the case where FOCL is instructed not to operationalize intensionally defined predicates.[8]

By using a uniform information gain metric, FOCL can deal with incomplete and incorrect domain theories. 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 likelihood of 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.

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 the assumption that rooks only attack in files, kings may move like knights, kings may attack anywhere in the same rank, and two pieces may occupy the same square provided the black king is not in an adjacent file. With these four errors, the domain theory correctly classified 76.2% of the examples when tested on 10,000 training examples. Appendix II provides an edited trace of FOCL operating with this domain theory. 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.

Table 9. Design of FOCL.

```
Let Pos be the positive tuples.
Let IR be the initial rule.
Let Body be empty.

Until Pos is empty
  Leg Neg be the negative tuples.
  Call LearnClauseBody.
  Remove from Pos those tuples covered by Body.
  Set Body to empty.

Procedure LearnClauseBody:
  If a ClauseBody of IR has positive gain,
    Choose best ClauseBody[1]
    Operationalize and delete superfluous literals from it,[2]
    Conjoin result with Body,
    Update Pos and Neg,
    Call ExtendBody,[3]
  Else
    Choose best literal,
    Operationalize and delete superfluous literals from it,[2]
    Conjoin result with body,
    Update Pos and Neg,
    Call LearnClauseBody.

Procedure ExtendBody:
  While Neg is non-empty
    Choose best literal.[2]
    Operationalize and delete superfluous literals from it.
    Conjoin result with Body.
    Update Pos and Neg.
```

[1]Takes advantage of good prior clauses.
[2]Allows use of non-operational predicates.
[3]Allows correction of old clause bodies.

Table 10. Domain errors.

Deleted Clause: (Theory loses coverage)
rook_attack_king(*R1*, *F1*, *R2*, *F2*, *F3*) ← *equal_rank*(*R2*, *R3*),
king_not_between_file(*R1*, *F1*, *R2*, *F2*, *F3*).

Added Clause: (Theory too general)
king_attack_king(*R1*, *F1*, *R2*, *F2*) ← *knight_move*(*R1*, *F1*, *R2*, *F2*).

Deleted Literal: (Clause too general)
Changed: *king_attack_king*(*R1*, *F1*, *R2*, *F2*) ← *equal_rank*(*R1*, *R2*), *adjacent_file*(*F1*, *F2*)
To: *king_attack_king*(*R1*, *F1*, *R2*, *F2*) ← *equal_rank*(*R1*, *R2*).

Added Literal: (Clause overly specific)
illegal(*R1*, *F1*, *R2*, *F2*, *R3*, *F3*) ← *same_loc*(*R1*, *F1*, *F2*, *F2*), *adjacent_file*(*F1*, *F3*).

Through a single mechanism, FOCL responds to each type of modified domain theory in a diferent manner. If these modifications result in negative gain, FOCL will not operationalize this rule but instead finds an accurate definition using as much of the domain

theory as possible and fills in the remainder with its bottom-up inductive method. In general, FOCL responds to the four types of correctness destroying transformations as follows:

- **Literal Deletion:** If the literals defining a clause have positive gain, then FOCL can operationalize the clauses that are not altered in a purely explanation-based fashion. If the clause with the literal deleted has positive gain, then FOCL operationalizes this clause and then uses inductive methods to complete the clause by finding a predicate that correctly classifies the remaining positive tuples and doesn't cover any negative tuples. Quite often, using induction to finish a clause results in finding the literal that was deleted.
- **Clause Deletion:** FOCL can operationalize the clauses that are not deleted. Operational descriptions equivalent to the remaining clauses (i.e., they cover the positive training tuples not covered by the remaining clauses and do not cover any negative tuples) are added in an inductive fashion.
- **Literal Addition:** Often, the unchanged clauses will have greater positive gain than the modified clause and the unchanged clauses will be operationalized first. At this point, the altered clause will not have positive gain, and an operational description equivalent to the clause before modification is added inductively. If an altered clause has positive gain, the greedy deletion of literals will usually remove the superfluous literal. If this fails, an overly specified clause is included in a concept definition. To cover additional positive tuples, FOCL inductively adds clauses.
- **Clause Addition:** FOCL operationalizes clauses with maximum gain and it is unlikely that randomly added clauses will have more gain. In effect, FOCL uses information gain as a method to find a subset of the domain theory that is accurate. FOCL iteratively finds an operational specialization with the highest positive gain on all training tuples and makes this the next clause. It removes those positive training tuples covered by that clause, and finds other operational clauses to cover the remaining positive tuples.

It is important to stress that FOCL does not contain any special code to deal with each type of modification. Rather, the above behavior falls out of using a uniform, information-based heuristic to judge the usefulness of operationalizing the target concept or using bottom-up methods to extend a clause.

In order to test the hypothesis that an incorrect domain theory will also increase the accuracy of FOCL, we ran an experiment in which we compared FOCL without a domain theory to FOCL with the above incorrect domain theory for *illegal*. Both versions of FOCL did not use typing. Figure 2 also plots the mean accuracy of FOCL with an incorrect domain theory. An analysis of variance indicates that even an incorrect domain theory has a significant effect on the accuracy of FOCL ($F(1,380) = 120.1, p < .0001$). The figure illustrates that FOCL learns more quickly with a domain theory that is only 76.2% accurate, than with no domain theory. This is a result of using the information-based heuristic to judge the usefulness of the domain theory and to switch between explanation-based and empirical methods.

5.1. Experiments on the size of the hypothesis space searched

In the previous section, we presented several experiments demonstrating that prior knowledge improves the accuracy of FOCL. The following experiments demonstrate that FOCL, using a combination of explanation-based and empirical learning methods, can advantageously use partial domain theories to reduce the size of the search space explored, even when these theories contain severe errors. Incorrect domain theories for the definition of *illegal* are generated from a correct theory by four different perturbation operators.

1. Randomly deleting a term from a clause of a rule (subject to the constraint that a clause must have at least one term). This modification will cause the rule to make errors on negative training examples.
2. Randomly deleting a clause from a rule (subject to the constraint that a rule must have at least one clause). This modification will cause the rule to make errors on positive training examples.
3. Randomly adding a term to a clause of a rule. A term was constructed randomly from the set of operational predicates and from the existing variables of a clause. This modification will cause the rule to make errors on positive training examples.
4. Randomly adding a clause to a rule. A clause was constructed with random terms. All clauses were at least 1 term long. There was a 0.5 probability 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 (641) 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 3 plots the accuracy of the resulting domain theory (averaged over 20 trials) and the number of literals tested by FOCL for each operation as a function of the number of modifications to the domain theory. Note that FOCL is able to exploit extremely inaccurate domain theories to constrain the search for a concept definition.

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 is 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 4 plots the accuracy of the domain theory and the number of literals expanded by FOCL with and without a 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 20 trials). The results of adding and deleting clauses and literals indicate that FOCL with an incorrect and incomplete domain theory explores a smaller portion of the search space than FOCL without a domain theory.

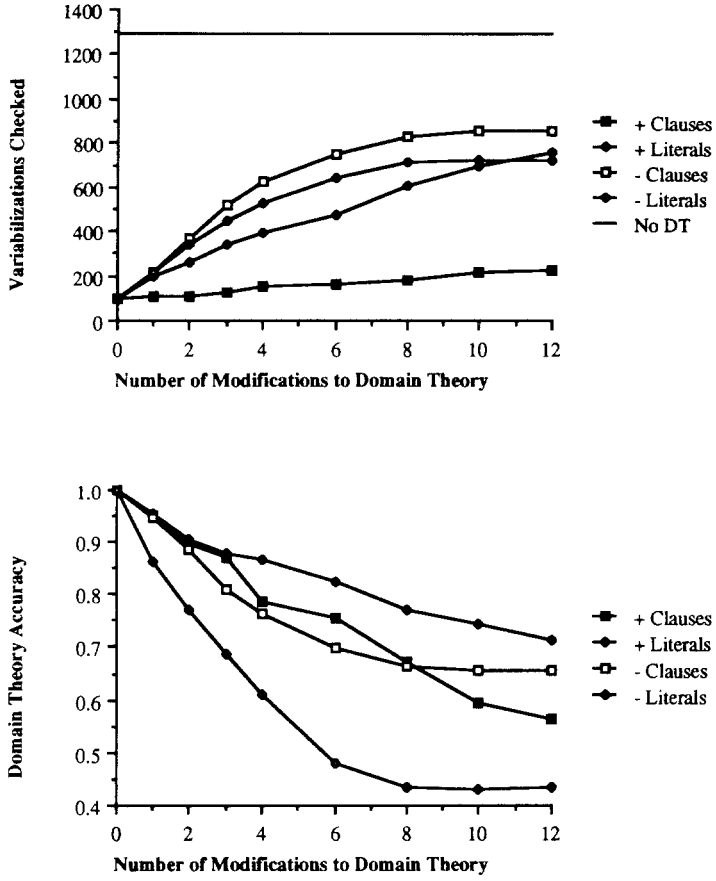

Figure 3. Upper: The effect of modifying a domain theory by individually adding literals, deleting literals, adding clauses and deleting clauses on the amount of search required by FOCL to learn an accurate concept. *Lower:* The accuracy of the modified domain theory.

5.2. Types of incomplete and incorrect domain theories

The accuracy of the domain theory is not the only important characteristic for predicting FOCL's ability to accurately learn a concept. In order for FOCL to tolerate incomplete and incorrect domain theories, the inductive process must be able to "patch" the operationalized conjunctions of literals derived from the domain theory. Since the inductive process does hill-climbing search guided by an information gain evaluation heuristic, there is no guarantee that the resulting clause will be best. For example, consider the chess domain where we mutate a few of the predicates. In particular, we replace the predicate *equal*(*X*, *Y*) with the predicate *half_eq*(*X*, *Y*), defined as {(1, 1), (2, 2), (3, 3), (4, 4), (7, 8)}. Similarly, we replace *adjacent* and *between* by extensional predicates that cover half of the

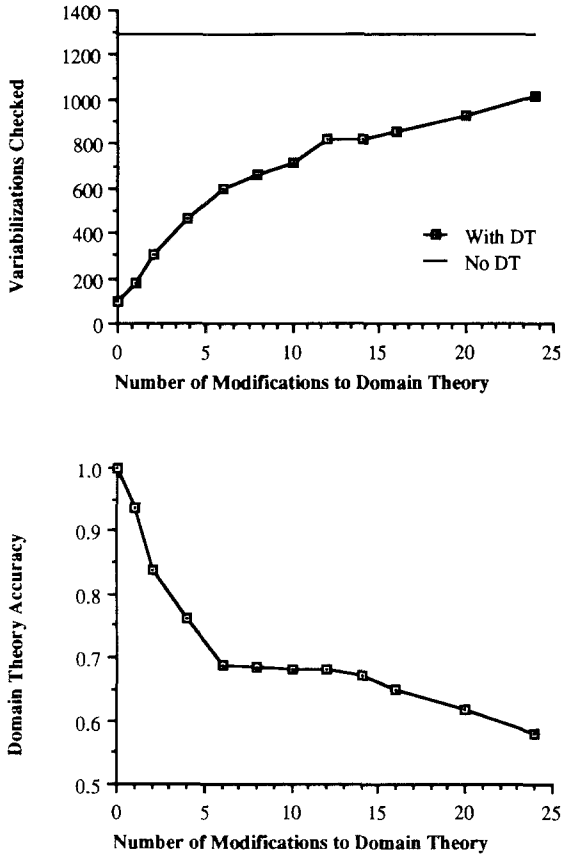

Figure 4. The effect of modifying a domain theory by combinations of adding literals, deleting literals, adding clauses and deleting clauses on the accuracy of the domain theory and on the amount of search required by FOCL.

positive examples of the proper predicates and a few negative examples. Since such a domain theory would have positive information gain, FOCL would create clauses by operationalization. For example, operationalizing *rook_attack_king* yields

$$illegal(A, B, C, D, E, F) :- half\_eq(E, C), not(half\_eq(C, A)).$$

Because this covers some negative examples, FOCL patches it inductively. Assuming the inductive learning can use the predicate *equal* (as well as *half_eq*), then the best possible patch is:

$$illegal(A, B, C, D, E, F) :- half\_eq(E, C), not(half\_eq(C, A)), equal(E, C), not(equal(C, A)).$$

However, this covers fewer positive examples than the ideal clause, so FOCL would have to learn another clause inductively to cover the remaining positive examples. The best clause is:

illegal(*A, B, C, D, E, F*): − *equal*(*E, C*), *not*(*equal*(*C, A*)).

In this case, a 78% correct domain theory does not help FOCL. Instead, it creates an arbitrary subdivision of the examples such that to achieve 100% accuracy, the inductive learner must learn the same literals from each set of positive examples. We ran FOCL with such an incorrect domain theory and with no domain theory on 25 trials of 40, 80, and 160 examples. In this experiment, the presence of a domain theory resulted in decreased accuracy ($F(1,144) = 9.11$, $p > .01$).

This experiment indicates that the accuracy of a domain theory is not the only factor that affects the accuracy of the learned concept. Rather, the domain theory must have the property that an accurate concept description can be achieved by inductively patching its operationalizations. The ability to patch the domain theory inductively depends on the predicates available for induction. In the examples in Section 4 and Section 5.1, the inductive component corrected for an imperfect domain theory by adding the literal that was deleted from a clause. Similarly, it could delete a literal that was added to the domain theory, ignore parts of the domain theory, or induce a clause that was deleted from the domain theory. However, in the above example, the only way to exclude some negative examples from a conjunction of literals formed by operationalization was to conjoin them with a set of literals formed by induction.[9] If the inductive learner had a predicate *not_7_and_8*(*X, Y*) that was true except when X = 7 and Y = 8, then it would be able to conjoin this predicate with *half_eq*(*X, Y*) to rule out negative examples: *illegal*(*A, B, C, D, E, F*): −*half_eq*(*E, C*), *not_7_and_8*(*E, C*), ...

We have explored a method for addressing this problem. As we have described FOCL so far, it operationalizes the domain theory provided it has positive gain. Another alternative is to compare the gain of the domain theory to the gain of literals formed solely by induction. On this problem, this variant of FOCL performed equally as well as FOCL with no domain theory. However, it achieves this accuracy by simply ignoring the domain theory. A better approach might be to consider a "paradigm shift" by replacing the extensional definition of *half_eq* with that of *equal* in the domain theory. Of course, this is much simpler if the definition of *equal* were known. On the other hand, a true paradigm shift often requires inventing such a new predicate.

5.3. Experiments on the prediction of the tuple size growth

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.[10] We trace the tuple size as each literal is selected and compare it with the tuple size predicted by equation 4 from Section 3.2. In order to apply this equation, we need the densities of all the predicates (see Table 11). The comparison of the predicted tuple size versus the actual tuple size is presented in Table 12.

Table 11. Predicate densities.

Predicate	Density
equal_rank	.125
not equal_rank	.875
adjacent_rank	.25
not adjacent_rank	.75
between_rank	.21875
not between_rank	.78125

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	5	5
†413	equal_rank	52	7
7	not between_file	5	5

All these results are from one entire episode of FOCL learning *illegal*. As Table 12 indicates, the estimates are very reasonable, giving empirical support to our earlier analysis. Our 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 predicted one. In these two cases, the sample is not a mixture of positive and negative cases, but nearly entirely negative. In these two cases, 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 nor overly general.

6.2. 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.

6.3. A-EBL

The A-EBL system (Cohen, in press) 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 EBL, 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 3). 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.

6.4. 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 *Branching Factor Depth*^{d}. 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 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 also use domain knowledge in inductive learning, by searching for non-operational predicate variabilizations.

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. With the exception of iterative widening, the other constraints on induction analyzed here could easily be incorporated with the pruning method of FOIL.

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 also easily be incorporated into FOIL to 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 relational 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 FOCL and the effect of different sorts of knowledge on this complexity. Some of these conclusions are summarized here:

- The branching factor grows exponentially in the arity of the predicate to be learned, in the maximum arity of the available predicates, and in the number of new variables required.
- The branching factor grows linearly in the number of available predicates.
- 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.
- Partially 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 potentially 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.
— 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.

In this paper, we have presented the core of FOCL. Our current research with FOCL includes dealing with noise in the context of relational learning (Brunk & Pazzani, 1991), revising incorrect domain theories (Pazzani & Brunk, 1990), exploiting knowledge of commonly occurring patterns of literals called relational clichés (Silverstein & Pazzani, 1991), allowing restricted function symbols and both nominal and ordinal constants into the representation language (Silverstein & Pazzani, 1991), exploring heuristics other than information gain or entropy such as Laplacian error estimators, creating new extensionally defined predicates, and including non-operational literals in learned concept descriptions.

Acknowledgments

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, William Cohen for discussions on types of imperfect domain theories, and Cliff Brunk, Tim Cain, Caroline Ehrlich, Ross Quinlan, Wendy Sarrett and Glenn Silverstein for reviewing a draft of this paper.

Notes

1. In the context of this paper, we use the term "domain theory" to refer to a set of (possibly incorrect and incomplete) Horn clauses that may be useful in defining the concept to be learned. The more general term, "prior knowledge" also includes information such as the types of predicates.
2. For some problems, this is not a severe representational restriction. For example, *color* (*X,red*) may be represented as *color*(*X, Y*), *red*(*Y*), although this representation can greatly increase the search cost.
3. 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).
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 assumed a 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.

5. Since FOCL first searches a smaller hypothesis space then FOIL, it is possible that it will produce a different answer than FOIL on the same data. We have not observed this on the experiments reported in this paper. However, when testing on noisy data (Brunk & Pazzani, 1991), we have observed that the larger hypothesis space results in more accurate concepts.
6. 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.
7. 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*) require finding all bindings of *G* such that *between*(1, *G*, 5) is true.
8. 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.
9. One could attempt to use a postprocessor to correct the output of FOCL. However, if the given predicates and domain theory are too misleading, then the sets of remaining positive and negative examples are smaller than they would be with no domain theory at all. Consequently, induction is more likely to make a mistake that cannot be corrected by postprocessing.
10. 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*)).
11. This recurrence was determined by Dan Hirschberg.

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.

Brunk, C., & Pazzani, M. (1991). An investigation of noise tolerant relational learning algorithms. *Proceedings of the Eighth International Workshop on Machine Learning* (pp. 389–393). Evanston, IL: Morgan Kaufmann.

Cohen, W. (in press). Abductive explanation-based learning: A solution to the multiple inconsistent explanation problem. *Machine Learning.*

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.

DeJong, G., & Mooney, R. (1986). Explanation-based learning: An alternate view. *Machine Learning, 1,* 145–176.

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). Inegrating 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 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., & Feng, C. (1990). Efficient induction of logic programs. *Proceedings of the First Conference on Algorithmic Learning Theory.* Tokyo, Japan: Ohmsha.

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 Systems.* Banff, Canada.

Pazzani, M., Brunk, C., & Silverstein, G. (1991). A knowledge-intensive approach to relational concept learning. *Proceedings of the Eighth International Workshop on Machine Learning* (pp. 432–436). Evanston, IL: Morgan Kaufmann.

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

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

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.

Silverstein, G., & Pazzani, M. (1991). Relational clichés: Constraining constructive induction during relational learning. *Proceedings of the Eighth International Workshop on Machine Learning* (pp. 203–207). Evanston, IL: 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. Detailed analysis of $v(m, k)$

In the text we approximated $v(m, k)$. Here we compute it exactly. To count the total number of distinct variabilizations, it is convenient to count the number of distinct variabilizations that use exactly j positions for old variables. We use $p(m, k, j)$ to represent this number. Since we require at least one old variable,

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

Now $p(m, k, j)$ is the product of the number of ways of picking j positions, the number of ways of assigning old variables to them, and number of ways of assigning the new variables to the remaining positions. The number of ways of picking j positions from m is $\binom{m}{j}$. The number of ways of filling these j positions with any of the k old variables is k^j. Now we have to count the number of ways to fill the remaining $m - j$ 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 variabilizations 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:[11]

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

To understand this recurrence, consider two cases. In case one, the last of the i positions is filled with a distinct new variable, giving rise to $new(i - 1, j - 1)$ distinct variabilizations. In case two, all of the j variables are already placed in the first $i - 1$ positions. This yields the second summand.

The boundary conditions are:

$$new(i, 0) = 0$$
$$new(i, 1) = 1$$
$$new(i, j) = 0, \text{ if } i < j$$

We define *anynew*(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

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

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

$$p(m, k, j) = \binom{m}{j} * k^j * anynew(m - j).$$

Summing this function as j ranges from 1 to m will give the total number of ways of variabilizing a predicate of arity m.

Appendix II. An annotated trace of FOCL

We present here an annotated trace of FOCL using the incorrect domain theory of *illegal* from Section 5. To create an operational definition for *illegal*, FOCL first computes the information gain of each given clause for *illegal*:

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

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:

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))

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.

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)

If the domain theory were 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:

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)	

Because some negative examples are covered by this clause, the clause is extended inductively. FOCL computes the information gain of every variabilization of every operational predicate (and its negation) and selects the literal with the maximum gain: *adjacent_file*(*F*3,*F*1).

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)

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 5

illegal(R1,F1,R2,F2,R3,F3):-equal-rank(R1,R3),equal-file(F1,F3)

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 6

illegal(R1,F1,R2,F2,R3,F3):-equal-file(F3,F2),equal-file(F1,F2)
not(between-rank(R2,R1,R3))

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.

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
BEST CONDITION not(equal-rank(R1,R2))		

CLAUSE 7

illegal(R1,F1,R2,F2,R3,F3):-equal-rank(R3,R2),not(equal-rank(R1,R2))

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 again, the clause with the maximum information gain is the clause with the deleted literal. This indicates that a board is in an illegal state if the two kings are in the same rank. Inductive techniques finish this clause by adding the restriction that the rook be in this rank, but not between the two kings. In effect, this clause had been extended to cover the case where the white rook is attacking the black king.

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

BEST condition not(between-file(F2,F1,F3)).
CLAUSE 8 illegal(R1,F1,R2,F2,R3,F3):-equal-rank(R1,R3),
equal-rank(R3,R2), not(between-file(F2,F1,F3)).

Finally, the last clause is learned entirely inductively and indicates that a board is in an illegal state if the white king and rook are in the same square.

CLAUSE 9
illegal(R1,F1,R2,F2,R3,F3):-equal-file(F2,F1),equal-rank(R2,R1)