An information-based approach to integrating empirical and explanation-based learning

Michael J. Pazzani
pazzani@ics.uci.edu
Clifford A. Brunk
brunk@ics.uci.edu
Glenn Silverstein
silverst@ics.uci.edu

Technical Report 91-38

April 25, 1991

We would like to thank Ross Quinlan for his advice on FOIL, Dennis Kibler for helping with a complexity analysis of FOIL, Kamal Ali for helping with the implementation, and Tim Cain and Caroline Ehrlich for commenting on an earlier draft of this paper.

1 Introduction

Recent research has shown that learning, like many other artificial intelligence problems, can be viewed as a knowledge-intensive activity. In this paper, we address two issues in knowledge-intensive learning:

- Taking advantage of domain knowledge that may be incomplete and incorrect. Rajamoney and DeJong (1987) call a domain theory "incomplete" if there are some positive examples of a concept that cannot be explained by the domain theory, and "incorrect" if there are some negative examples that are explained as positive examples by the domain theory.
- Learning when there is noise in the training data. Here, we focus on learning when there is the possibility that training data is incorrectly classified.

These problems are particularly important since their solution will increase the class of problems that can be addressed by knowledge-intensive learning algorithms. Many applications have training data that are incorrectly classified. Furthermore, there are many problems in which the encoding of the domain knowledge results in some form of incompleteness and incorrectness. Explanation-based learning (DeJong & Mooney, 1986; Mitchell, Keller, & Kedar-Cabelli, 1986), an initial formulation of knowledge-intensive learning, did not address either of these issues.

More recently, a variety of systems have integrated explanation-based learning with some form of empirical learning (e.g., Ali, 1989; Bergadano & Giordana, 1988; Danyluk, 1989; Fawcett, 1989; Flann & Dietterich, 1989; Hirsh, 1989; Lebowitz, 1986; Michalski & Ko, 1988; Mooney & Ourston, 1989; Ourston & Mooney. 1990; Pazzani, 1990; Shavlik & Towell, 1989; Van Lehn, 1987; Wilkins & Tan, 1989). However, none of these systems provide a systematic analysis of the effect of noise in the training data.[1] In addition, some of the systems can deal with domain theories that are incomplete, but not incorrect, while others deal with domain theories that are incorrect, but not incomplete. Furthermore, due to limitations of the empirical learning algorithm they use, most of the systems require that the domain theory be represented in propositional logic, a language that is less expressive than the Horn-clause representations typically used to represent domain theories for EBL.

We present an integrated learning system whose empirical component is based on FOIL (Quinlan, 1990), a recent advance in Horn-clause learning and whose explanation-based component has been designed to work in a manner compatible with FOIL. In particular, the proof and operationalization process used by the explanation-based component is guided by the same information-based metric used to select hypotheses in the empirical component.

In the remainder of this paper, we first review the FOIL algorithm. Next, we introduce FOCL (First-Order Combined Learner). Finally, we report on a series of experiments in which we test FOCL by supplying it with incorrect and incomplete domain theories (formed by mutating a correct domain theory) and by introducing noise into training examples. The experiments are centered around two problem domains. The first problem is to determine if a chess board containing a white king, white rook, and black king is in an illegal board configuration.[2] The second problem domain involves determining if a student is required to pay back a student loan based on student enrollment and background information. In these experiments we demonstrate that providing domain knowledge to FOCL can decrease the amount of search required during learning and increase the accuracy of learned concepts even when the domain knowledge is incorrect and incomplete and there is noise in the training data.

2 FOIL

FOIL learns constant-free Horn-clause theories that serve as intensional definitions of a concept, P_o. The definition of a concept consists of a set of *clauses* of the form:

$$P_0(V_{0,1},\ldots, V_{0,n_0}) :- P_1(V_{1,1},\ldots, V_{1,n_1}) , \ldots ,P_m(V_{m,1},\ldots, V_{m,n_m}).$$

where each clause represents an alternative method of proving that an example is instance of P_o. The clauses consist of a conjunction of *literals,* which are each composed of a particular predicate (e.g., P_i) and an ordering of variables for the predicate (e.g., $V_{i,1},\ldots, V_{i,n_i}$). The variables of the literal are classified

1. One exception is described in Shavlik and Towell (1989). Their system has the potential of handling incomplete and incorrect domain theories and noisy data. However, the system requires that domain theories be represented in propositional logic so that they can be transformed in neural networks.

2. A board configuration is illegal if either king is in check or more than one piece occupies the same space.

as new and old as follows: a variable of a literal is called "new" if it does not appear in the head of the current clause or in any literal to the left of the current literal; otherwise, the variable is called "old."

An example in FOIL is represented as a *tuple*, which contains values for the variables of the predicates to be learned. For example, when learning the definition of `illegal(A,B,C,D,E,F)`, A and B are the position of the white king (i.e., the rank and file which are represented by a number between 1 and 8), C and D are the white rook's position, and E and F are the black king's position. Training examples for this problem would consist of a set of 6-tuples whose elements correspond to the ranks and files of the three pieces. Each tuple is identified as a positive or negative instance of `illegal(A,B,C,D,E,F)`. (1,2,1,5,1,3) would be a positive instance of `illegal(A,B,C,D,E,F)`, since the kings at 1,2 and 1,3 are in check.

FOIL also takes as input a set of predicates $\{P_1, ..., P_n\}$ which are extensionally defined. For example, `adjacent(X,Y)` is defined by the set `{(1,2)(2,1)(2,3)(3,2)(3,4)(4,3)(4,5)(5,4)(5,6)(6,5)(6,7)(7,6)(7,8)(8,7)}`. These predicates can be used to form the literals that make up the clauses for P_0. For example, the predicates `between(X,Y,Z)` (`Y` is greater than `X` and less than `Z`), `equal(X,Y)` (`Y` is equal to `X`) and `adjacent(X,Y)` (`Y` is either one less than or one greater than `X`) are useful in learning illegal. In FOIL, these predicates must be defined extensionally.

In effect, FOIL has two operators: start a new empty clause, and add a literal to the end of the current clause. FOIL performs the second operator until no negative examples are covered by the clause, and performs the first operator adding new clauses until all positive examples are covered by some clause. FOIL computes the information gain of the legal *variabilizations*[3] of each extensionally defined predicate in order to determine which literal to add to the end of a clause. A variabilization is a particular ordering of new and old variables. The information gain of the addition of a new literal to the current clause is defined as follows:

$$\text{Gain(Literal)} = T^{++} * \log_2(p_1/p_1+n_1)-\log_2(p_0/p_0+n_0)$$

where p_0 and n_0 are the current number of positive and negative tuples, p_1 and n_1 are the number of positive and negative tuples that would remain after adding the literal, and T^{++} is the number of current positive tuples that have at least one corresponding tuple[4] in the positive tuples after adding the literal (Quinlan, 1990).

Table 1 presents an overview of the FOIL algorithm. Pazzani and Kibler (1990) argue that the number of times that the information gain of a literal is computed is a good metric for indicating the size of the search space explored by FOIL.

An additional feature of FOIL is that it contains a stopping criteria for deciding whether there is sufficient data to support adding a literal to a clause (or creating a new clause). The stopping condition compares the number of bits needed to explicitly encode the data[5] to the number of bits required to encode the new literal.[6] For brevity, this feature of FOIL was not included in Table 1, which should read "`Until Pos (or Neg) is almost empty, as determined by the stopping criteria.`"

3 FOCL

FOCL extends FOIL in a variety of ways to take advantage of domain knowledge. Pazzani and Kibler (1990) describe how adding knowledge about the extensional predicates (e.g., the types of variables, and information about commutativity of predicates) can be used to reduce the search space. Here, we concentrate on how adding knowledge in the form of a domain theory can increase the accuracy of concepts learned and decrease the search space explored. We first describe how intensionally defined predicates can be used by FOCL and how they are operationalized. Finally, we discuss how providing a goal concept can limit the amount of search used by FOIL.

3. A legal variabilization must include at least one old variable, and not cause infinite recursion (Quinlan, 1990).

4. Note that the size of the tuples may grow in FOIL when a literal introduces new variables. For example, when learning `illegal(A,B,C,D,E,F)`, if the first literal selected is `between(A,G,C)`, then for the remainder of the clause the 6-tuples used will be extended to 7-tuples by adding those values of `G` for which `between(A,G,C)` is true. Furthermore, not every 6-tuple may have a corresponding 7-tuple in the extended set, and some 6-tuples may have more than extension in the new set of 7-tuples.

5. given by $\log_2(p+n)+\log_2\left(\binom{p+n}{p}\right)$ where p is the number of positive examples and n is the number of negative examples.

6. given by $1+\log_2(r)+\log_2(a_p)$ where r is the number of predicates and a_p is the number of possible variabilization of the predicate.

Table 1. An Overview of FOIL

```
Input
    Pred:   Name of the predicate to learn
    Vars:   An ordered tuple of variable names for the predicate
    Pos:    A set of tuples for the positive examples of the predicate
    Negs:   A set of tuples for the negative examples of the predicate
    Preds:  A set of extensionally defined predicates

Set Clauses to empty
Until Pos is empty (
        Set NewClause to empty
        Set Old to Vars
        Until Negs is empty (
              For each Predicate in Preds (
              For each V in variabilizations(Pred,Old) (
                        create a Literal from Predicate and V
                        compute_gain(Literal,Pos,Neg) ))
                Conjoin the literal with the maximum gain to NewClause
                Add any new variables in the literal to Old
                Set Pos to the extensions of Pos satisfied by Literal
                Set Neg to the extensions of Neg satisfied by Literal )
        Remove from Pos all tuples that satisfy the NewClause
        Reset Negs to the original negative tuples
        Add NewClause to Clauses )
```

3.1 Intensionally defined predicates

In order to compute the information gain of a literal, it is necessary to count the number of tuples that would result from adding the literal to the current clause. If the literal is formed from a predicate that is defined extensionally (as in FOIL), this can be accomplished by a relational join operator. For example, if the current set of positive tuples consists of the `(X,Y)` pairs: `((2,1)(3,1)(5,2))` and the predicate `between` is defined extensionally as: `((1,2,3)(1,2,4)(1,2,5)(1,3,4)(1,3,5)(2,3,4)(2,3,5)(2,4,5)(3,4,5))` then the literal `between(Y,Z,X)` would extend the set of positive tuples to include all values of `Z` such that `between(Y,Z,X)` is true. This produces `((3,1,2)(5,2,3)(5,2,4))` as the new set of positive tuples. The first tuple, `(3,1,2)` is added since `2` is the only value for which `between(1,Z,3)` is true. The other two tuples are possible values for `between(2,Z,5)`. There is no value of `Z` for which `between(1,Z,2)` is true, so there are no extensions of this tuple that are included in the new set of tuples.

In FOCL, predicates may also be defined intensionally (as in Prolog rules). The intensionally defined predicates are defined by rules that indicate how the predicate may be inferred from extensionally defined predicates (or other intensionally defined predicates). For example, `between` could be defined in terms of `less_than` as follows:

```
between(A,B,C) :-less_than(A,B), less_than(B,C).
```

where `less_than` is defined extensionally as: `((1,2)(1,3)(1,4)(1,5)(2,3)(2,4)(2,5)(3,4)(3,5)(4,5))`. To compute the information gain of intensionally defined predicates such as `between`, FOCL makes use of a backward chaining proof procedure similar to the prolog predicate `setof`. For example, the Prolog goal `setof((5,2,Z),between(2,5,Z),T)` will bind `T` to those extensions of `(5,2)` for which `between(2,Z,5)` is true. FOCL computes the information gain of the literals formed from an intensionally defined predicate by computing the size of the sets of positive and negative examples that arise from adding the literal to the current clause using a mechanism similar to `setof`.[7] As with the extensionally defined predicates, FOCL computes the information gain of every variabilization of the intensionally defined predicates in an attempt to find a literal (i.e., a predicate and associated variabilization) which maximizes information gain. Furthermore, it is possible to directly compare the informational gain of literals formed from intentionally defined predicates to the literals formed from extensionally defined predicates to find the literal with the maximum overall information gain.

7. FOCL is implemented in Common Lisp and makes use of a backward chaining rule interpreter whose semantics are identical to that of pure Prolog. This allows us to take advantage of Prolog's expressiveness when appropriate, while having control over the unification and indexing used by the interpreter.

Table 2. The operationalization process in FOCL.

```
operationalize(Literal, Pos, Neg):
    Initialize Conjuncts to the empty set
    For each Clause in the definition of Literal
        compute_gain(Clause, Pos, Neg)
    Let Clause be the clause with the maximum gain
    For each literal  L in Clause
        if L is operational
            then Set Pos to the extensions of Pos satisfied by L
                 Set Neg to the extensions of Neg satisfied by L
                 Add L to Conjuncts
            else add operationalize(L, Pos, Neg) to Conjuncts
    Return Conjuncts
```

3.2 Operationalization

If the literal which maximizes information gain is formed from an intensionally defined predicate, it is possible for FOCL to add this literal to a clause. However, typically we run FOCL in a mode in which it operationalizes (Keller, 1987) such a literal, by expressing it in terms of extensionally defined predicates. Note that in FOCL, we consider the terms "operational" and "extensionally defined" to be equivalent. Similarly, we equate "non-operational" with "intensionally defined."

Unlike EBL, the operationalization process in FOCL uses a set of positive and negative examples, rather than a single positive example. A non-operational literal is operationalized by producing an specialization that is a conjunction of operational literals. When there are several ways of operationalizing a literal (i.e., there are multiple, disjunctive clauses), the information gain metric is used to determine which clause should be used. The same backward chaining proof process that computes the information gain of literals formed from an intensionally defined predicates is used to compute the information gain of a clause (since a clause is simply a conjunction of literals). Table 2 summarizes the operationalization process. The goal is to find the conjunction of operational literals with the maximum information gain.See Section 3.5 for an example and Section 4 for experimental results.

Providing domain knowledge in the form of intensionally defined predicates supplies an important form of guidance to the learning process. In particular, it is possible that a conjunction of extensionally defined literals will have positive information gain, while the individual literals do not. Such a conjunction cannot be found by hill-climbing search. For example, on 10 trials of 1000 training examples, FOCL was unable to obtain 100% accuracy when learning `illegal` in terms of the extensionally defined predicates `less_than`, `adjacent` and `equal`. However, when the intesionally defined predicate `between` was added, FOCL learned a 100% accurate definition of `illegal` in terms of the extensionally defined predicates in all 10 trials.

Unfortunately, there is also a disadvantage of adding intensionally defined predicates. In particular, each additional predicate added increases the amount of search performed during learning. If the arity of all intensionally defined predicates is no greater than the maximum arity of the extensionally defined predicates, then search is increased by at most a linear factor. However, if the arity of any intensionally defined predicate is greater than that of all extensionally defined predicates, search is increased by an exponential amount. This occurs because the size of the search space explored is dominated by the predicate with the largest number of arguments (Pazzani & Kibler, 1990). In any event, adding intensionally defined predicates in this manner has the undesirable consequence that as the number of predicates the system knows (or learns) increases, the amount of work required to learn a new predicate increases. In the next section, we address this problem.

3.3 Goal concepts

The problem of intensionally defined predicates increasing the search space occurs because in addition to checking the variabilizations of the extensionally defined predicates, the variabilizations of the intensionally defined predicates must be checked as well. For example, consider the definition of `same_loc` from the illegal domain theory in Table 3. Since there are a minimum of 6 old variables when learning `illegal`, and `same_loc` has 4 arguments, there are 29,168 distinct variabilizations of `same_loc`. However, only 3 of these variabilizations make sense in this domain. In particular, if `(R1,F1,R2,F2,R3,F3)` represent the rank and file of the white king, white rook and black king, then same_loc(R1,F1,R2,F2) determines if the white king and white rook occupy the same square. Similarly, same_loc(R1,F1,R3,F3) determines if the kings are on the same square and same_loc(R2,F2,R3,F3) determines if the white rook and black king are on the same square. It is possible that same_loc(R3,R2,R1,N1) will have some information gain when tested. However, this variabilization does not have a very meaningful interpretation in this domain and testing it is not likely to yield maximum information gain. Even if this variabilization has the maximum information gain, it is unlikely that the domain knowledge was intended to have this interpretation.

Goals concepts can be used to concentrate FOCL's search on the relevant variabilizations of the intensionally defined predicates. Mitchell, Keller, and Kedar-Cabelli (1986) call a non-operational definition of the concept to be learned a *goal concept*. Notice that the domain theory contains a non-operational (i.e., intensional) definition of the predicate illegal. The clauses for illegal indicate which variabilization of which predicates should be tested to create an operational (i.e., extensional) definition of illegal. When a goal concept is provided to FOCL, it computes the information gain of the goal concept. If the goal concept has positive information gain, FOCL forms a conjunction of literals to add to the current clause by operationalizing the goal concept. Otherwise, it computes the information gain of all variabilizations of all predicates. [8]

A goal concept may be operationalized in more than one way to produce separate clauses. When the goal concept has been operationalized and the clause has been completed (i.e., it excludes all negative examples), the set of positive tuples accounted for by a clause are removed. If the goal concept still has information gain, it is operationalized again in the next clause. A different operationalization must occur since the positive tuples covered by the prior operationalization are removed, and that path through the proof tree will not cover any of the remaining positive tuples. If any negative tuples are also accounted for by an operationalization, additional literals are added to the clause by induction until no negative examples are covered.

There is no fixed order in which empirical and explanation-based learning are performed. Instead, a uniform application of the information gain metric determines whether a conjunction of extensional literals is added to the current clause by operationalizing the goal concept, a single extensional literal is added by checking all variabilizations of the extensionally defined predicates, or a conjunction of literals is added by operationalizing a variabilization of one of the intensionally defined predicates. It can occur that the first literal of a clause must be learned empirically (i.e., by checking all variabilizations of the extensionally defined predicates) before the goal concept may have positive information gain. The next addition to the clause would be the conjunction of literals formed by operationalizing the goal concept. The final literals might be added empirically by operationalizing an intensionally defined predicate.

Table 3. illegal domain theory.

```
illegal(R1,F1,R2,F2,R3,F3) :- same_loc(R1,F1,R2,F2).
illegal(R1,F1,R2,F2,R3,F3) :- same_loc(R1,F1,R3,F3).
illegal(R1,F1,R2,F2,R3,F3) :- same_loc(R2,F2,R3,F3).
illegal(R1,F1,R2,F2,R3,F3) :- king_attack_king(R1,F1,R3,F3).
illegal(R1,F1,R2,F2,R3,F3) :- rook_attack_king(R1,F1,R2,F2,R3,F3).
same_loc(R1,F1,R2,F2) :-  equal(R1,R2), equal(F1,F2).
king_attack_king(R1,F1,R2,F2) :-  adjacent(R1,R2), adjacent(F1,F2)
king_attack_king(R1,F1,R2,F2) :-  adjacent(R1,R2), equal(F1,F2)
king_attack_king(R1,F1,R2,F2) :-  equal(R1,R2), adjacent(F1,F2)
rook_attack_king(R1,F1,R2,F2,R3,F3):- equal(R2,R3), king_not_between(R1,F1,R2,F2,F3).
rook_attack_king(R1,F1,R2,F2,F3,R3):- equal(F2,F3),king_not_between(F1,R1,F2,R2,R3).
king_not_between(X1,Y1,X2,Y2,Y3) :-  not(equal(X1,X2)).
king_not_between(X1,F1,X2,Y2,Y3) :-  equal(X1,X2),not(between(Y2,Y1,Y3)).
between(X,Y,Z) :- less_than(X,Y),less_than(Y,Z).
```

8. Actually, FOCL operates in two modes. The mode describe here is *theory mode*. In *information mode*, FOCL compares the information gain of the goal concept to the maximum information gain of literals formed from extensionally defined predicates, and adds the literal (or operationalized literals) with maximum information gain. In our experience, the mode does not have a significant effect on the accuracy of the hypotheses produced by FOCL unless the domain theory is extremely inaccurate (i.e., less than 60% accurate), in which information mode results in more accurate hypotheses. When the domain theory is more accurate, theory-mode results in less search than information-mode. When the domain theory is less accurate, theory-mode results in more search because the operationalized concepts tend to be overly specialized and more clauses are needed to cover the training examples. All experiments reported in this paper will be done in theory-mode.

3.4 Selective operationalization

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 means of finding 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 Section 4.3, we provide experimental evidence on the ability of the greedy technique to approximate the optimal solution.

3.5 Learning in spite of incorrect and incomplete domain theories: An example

Table 4 displays a domain theory for a problem involving repayment of student loans. Four errors were deliberately introduced into a correct domain theory (deleting a clause, deleting a literal from a clause, adding a new clause, and adding a literal to a clause). In addition, 50 training examples (25 positive, 25 negative) were used in this illustration. One of the positive examples is represented by the following operational predicates: `longest_absence_from_school(mary,3)`, `enrolled(mary, ucla,5)`, and`(disabled,mary)` and one of the negative examples is represented by `enrolled(bob,uci,10)`, `male(bob)` and `longest_absence_from_school(bob,12)`. Note that `mary` is erroneously classified by the domain theory as a negative example (because the rule for disability deferment has been modified by adding an extra condition) and `bob` is incorrectly classified as a positive example (because an extra clause has been added that states that students enrolled at UCI are eligible for a financial deferment).

To solve this problem, FOCL tries to operationalize the concept `no_payment_due`. There are two clauses that can be used to prove that no payment is due. FOCL computes the information gain of both and selects the alternative with the highest information gain. The predicate `eligible_for_deferment` is true of 16 positive and no negative examples (information gain = 16.0) and the predicate `continuously_enrolled` is true of 15 positive and 14 negative examples (information gain = 0.7). Hence, the predicate `eligible_for_deferment` is selected to be operationalized. There are five alternative ways of proving `eligible_for_deferment`. `military_deferment` has the highest information gain. Since there is only one way to operationalize `military_deferment` and its definition is already operational, the conjunction of literals `enlist(A, B) & armed_forces(B)` is used to start the first clause for `no_payment_due(A)`. No negative examples are covered by this operationalization, so the

Table 4. Domain theory for repayment of student loans.

Clauses and literals that were deleted to deliberately introduce errors are ~~stricken~~. Clauses and literals that were added are shown in **bold**.

```
no_payment_due(S) :- continuously_enrolled(S).
no_payment_due(S) :- eligible_for_deferment(S).
continuously_enrolled(S) :-  ~~never_left_school(S)~~,enrolled_in_more_than_five_units(S).
eligible_for_deferment(S) :- military_deferment(S)
eligible_for_deferment(S) :- peace_corps_deferment(S).
eligible_for_deferment(S) :- financial_deferment(S).
eligible_for_deferment(S) :- student_deferment(S).
eligible_for_deferment(S) :- disability_deferment(S).
military_deferment(S) :-  enlist(S,A),armed_forces(A).
peace_corps_deferment(S) :-  enlist(S,A),peace_corps(A).
financial_deferment(S) :- filed_for_bankruptcy(S).
~~financial_deferment(S) :- unemployed(S).~~
**financial_deferment(S) :-  enrolled(S,C,U),uci(C).**
student_deferment(S):-enrolled_in_more_than_eleven_units(S).
disability_deferment(S) :- **male(S)**,disabled(S).
never_left_school(S) :-  longest_absence_from_school(s,a),6>A.
enrolled_in_more_than_N_units(S,N) :-  enrolled(S,SCH,U),school(SCH),U>N.
```

process of building the first clause terminates. The second, third and fourth clauses of `no_payment_due` are formed by a similar manner, operationalizing `financial_deferment`, `peace_corps_deferment`, and `student_deferment`.

FOCL starts the fifth clause in a similar fashion. The predicate `eligible_for_deferment` is true of two positive examples and no negative examples, and has the maximum information gain. The clause `disability_deferment` is true of the two positive examples and no negative examples, and is operationalized to produce the conjunction `male(A) & disabled(A)`. FOCL then tries to improve the information gain of this conjunction by deleting literals from it as described in the previous section. When the literal `male(A)` is deleted, the information gain is increased since `disabled(A)` covers four positive examples and no negative examples. Therefore, the fifth clause is `no_payment_due(A) :- disabled(A)`.

For the sixth clause, there are no remaining positive examples explained by `eligible_for_deferment`. However, `continuously_enrolled` explains four of the six remaining positive examples and fourteen of the twenty-five negative examples, so it is operationalized and the conjunction `enrolled(A,B, C),school(B),C>5` is added as the first part of the sixth clause. Since the conjunction of literals formed by operationalizing `continuously_enrolled` covers some negative examples, FOCL tries to induce other literals that exclude these negative examples, and satisfies at least some of the positive examples. The non-operational predicate, `never_left_school`, has the highest information gain of the available predicates. Because this predicate is not operational, it too is operationalized and the resulting literals are conjoined with the operationalization of `continuously_enrolled` to produce the conjunction: `enrolled(A,B,C),school(B),C>5,longest_absence_from_school(A, D), 6>D` which excludes the remaining negative examples. Note that the first part of this clause was formed by EBL (operationalizing a goal concept with positive information gain) and the second part was formed inductively (by searching all variabilizations of all intensionally and all extensionally defined predicates).

At this point, the goal concept no longer has positive information gain. This occurs because the set of positive examples has been reduced by eliminating those positive examples that are satisfied by each new clause created. Since the goal concept can no longer correctly classify the remaining positive examples, FOCL must rely on inductive techniques to complete the definition. FOCL induces that unemployed persons are not required to make loan payments. At this point, all positive examples are covered by some clause, and no negative examples are covered by any clause, so the learning process terminates.

In this example, FOCL first operationalized as much of the domain theory as possible. Note that the first few clauses did not require any induction. Later clauses operationalized part of the domain theory, but used induction to add extra literals. This is a sign that the domain theory is close to being correct. Finally, FOCL used only inductive techniques to learn the final clause. This behavior is common and is a consequence of using information gain as a metric to guide a greedy search for a complete set of clauses that covers all positive and no negative examples.

4 Experimental Evaluation of FOCL

In this section, we report on three experiments with FOCL. These experiments are designed to test the following hypotheses:

- FOCL learning with domain theories that are incomplete and incorrect performs less work and produces more accurate concepts than FOCL learning with no domain theory.
- An incomplete and incorrect domain theory allows FOCL to tolerate classification noise better than no domain theory.
- The greedy algorithm for deleting literals from clauses is an efficient approximation of the optimal algorithm.

4.1 Learning with incomplete and incorrect domain theories

The following experiments are intended to determine whether FOCL's combination of explanation-based and empirical learning methods performs less work and produces more accurate concepts when learning is performed using an incomplete and incorrect domain theory than FOCL with no domain theory. Errors are introduced into the domain theory for `illegal` by using the following four operators:

- Randomly deleting a literal from a clause of a rule. This modification will cause the rule to make errors on negative training examples.
- Randomly deleting a clause from a rule. This modification will cause the rule to make errors on positive training examples.
- Randomly adding a literal to a clause of a rule. The added literal is constructed randomly from the set of operational predicates and from the existing variables of the current clause. This modification will cause the rule to make errors on positive training examples.
- Randomly adding a clause to a rule. The added clause is constructed with random literals. All clauses are at least 1 literal long and there is a 0.5 probability that clauses will have at least 2 literals, a .25 probability of at least 3, etc. This modification will cause the rule to make errors on negative training examples.

The design of this experiment follows a 2 algorithm (purely empirical vs. combined empirical and explanation) × 12 modification (0, 1, 2, 4, 6, 8, 10, 12, 14, 16, 20 and 24 modifications to the domain theory) design. The dependent variables measured were the accuracy of the learned concept and the number of times the information gain of a literal is computed. We ran 20 trials of FOCL with and without the domain theory. On each trial, 40 positive and 40 negative examples of illegal were randomly generated. Next, we randomly introduced a number of errors into the domain theory. Each operator had a .25 probability of being selected. We then ran FOCL with the modified domain theory and FOCL with no domain theory, and recorded the accuracy of the concept learned by each version (calculated by testing on 500 positive and 500 negative examples) and the number of times the information gain of a literal was computed. Figure 1 plots the accuracy of FOCL (top), the amount of work performed during learning (middle) and the accuracy of the modified domain theory used by FOCL (bottom--calculated by using the domain theory to classify 500 positive and 500 negative examples).

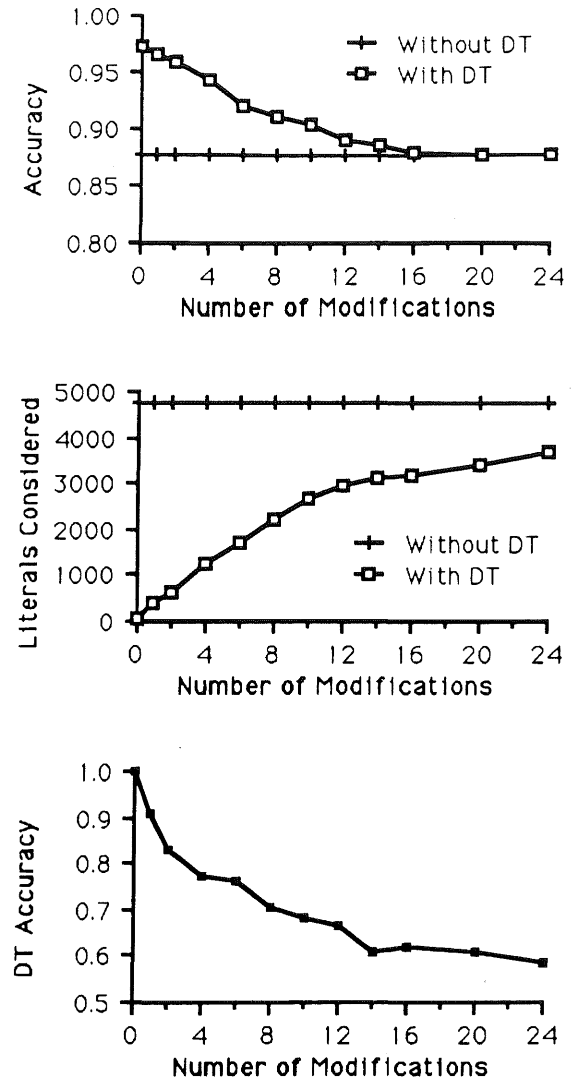

Figure 1. Accuracy of FOCL with and without a domain, number of literals tested during learning, and accuracy of domain as a function of the number of modifications randomly made to the domain theory.

The figure shows that FOCL with an incomplete and incorrect domain theory is at least as accurate as FOCL with no domain theory (even when the domain theory is less than 70% accurate). Furthermore, FOCL does less work when using an incomplete and incorrect theory than when using no domain theory. Of course, it is more expensive to compute the information gain of an intensionally defined predicate (when using a domain theory) than an extensionally defined predicate (when using only empirical methods). Nonetheless, there is also a savings in CPU time even when using an incorrect domain theory. For example, with an incomplete and incorrect domain theory formed by applying eight mutations to a correct theory, FOCL takes 21.1 CPU seconds while FOCL with no domain theory takes 30.4 CPU seconds (on a Sun 4). An analysis of variance indicates that, for accuracy, there was a main effect for the algorithm variable F(1,456) = 83.1 (p < .0001). This shows that the accuracy of FOCL with a domain theory is significantly different than the accuracy of FOCL with no domain theory. In addition, there was an interaction between the algorithm and the number of modifications F(1,11) = 6.47 (p <.0001). For the dependent variable representing the number of times the information gain of a literal is computed, there was also a main effect for the algorithm F(1,456) = 792.0 (p <.0001) and an interaction between the algorithm and the number of modifications F(1,11) = 14.6 (p <.0001).

In order to gain insight into how FOCL deals with each type of error in the domain theory, we also ran each operator separately. Figure 2 plots the accuracy of FOCL (top), the amount of work performed during learning (middle), and the accuracy of the modified domain theory used by FOCL (bottom). As in Figure 1, the dependent variables are averaged over 20 trials.

Through a single mechanism, FOCL responds to each type of modified domain theory in a different manner. If these modifications produce a rule in the domain theory with negative gain, FOCL will not operationalize this rule and instead builds a concept definition using the parts of the domain theory that have positive gain and fills in the remainder with its empirical method. In general, FOCL responds to the four types of modifications as follows:

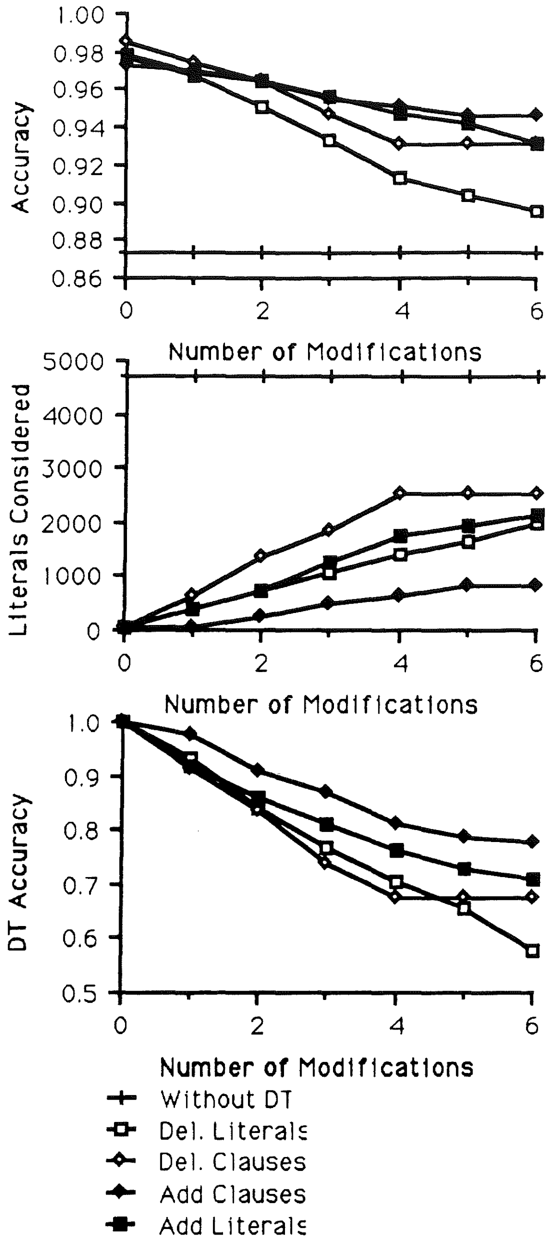

Figure 2. Accuracy of FOCl with and without a domain theory for each type of perturbation to the domain theory, number of literals tested during learning, and accuracy of domain theory as a function of the number of modifications randomly made to the domain theory.

- **Literal Deletion:** If the clause with the literal deleted has positive gain, then FOCL can operationalize this clause and then use empirical methods to complete the clause by finding a literal that correctly classifies some remaining positive tuples and doesn't cover any negative tuples.
- **Clause Deletion:** FOCL operationalizes the clauses that are not deleted. If these clauses do not cover the positive tuples covered by the deleted clause, then a clause equivalent to the operationalization of

the deleted clause can be learned empirically.

- **Literal Addition:** If the altered clause has positive gain, FOCL will operationalize it and attempt to delete literals to increase the information gain of the clause. If the altered clause has negative information gain, empirical learning is used to learn literals equivalent to the operationalization of the clause.
- **Clause Addition:** FOCL operationalizes clauses with maximum gain and it is unlikely that randomly added clauses will have more gain. Hence, this clause is likely to be ignored. FOCL serially finds operational specializations with the highest positive gain on all remaining tuples.

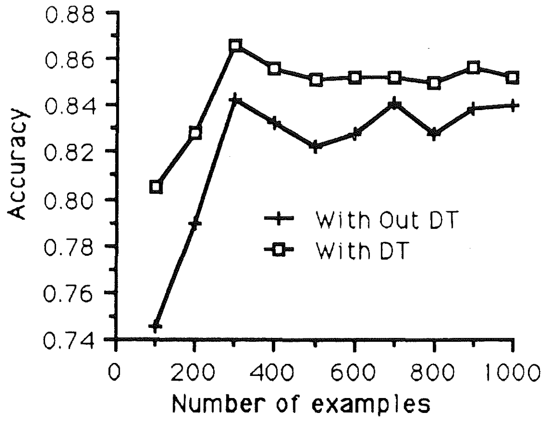

Figure 3. Accuracy of FOCl with and without a domain theory when there is 10% noise in training data and a 70.2% accurate domain theory.

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 goal concept are chosen in a greedy manner to cover the positive examples and exclude the negative examples. The more difficult problems for FOCL occur when the empirical component of FOCL is required to make up for domain theory errors. The empirical method is needed when no subset of the possible operationalizations of the domain theory will result in a correct hypothesis.

4.2 Learning from noisy data with incomplete and incorrect domain theories

In order to test the hypothesis that an incomplete and incorrect domain theory allows FOCL to tolerate classification noise better than no domain theory, we generated training sets of the `illegal` concept in which 10% of the data was assigned to a random class (rather than the correct class). In addition, we created one domain theory that was 70.2% accurate by using the mutation operators of the previous section. The design of this experiment follows a 2 algorithm (purely empirical vs. combined empirical and explanation) × 10 training set size (100, 200, ..., 1000) design. The dependent variable measured was the accuracy of the learning algorithm. Figure 3 shows the accuracy of FOCL with and without a domain theory as a function of the size of the training set. An analysis of variance indicates that there was a main effect for the algorithm variable $F(1,180) = 33.4$ ($p < .0001$). This shows that even when 10% noise is introduced into the training data, and the domain theory is 70% accurate, the accuracy of FOCL with a domain theory is significantly different than the accuracy of FOCL with no domain theory.

4.3 Greedy deletion of literals

To test the hypothesis that the greedy algorithm for deleting literals efficiently approximates the optimal algorithm, an experiment was run using FOCL with each of the two deletion algorithms. The design of this experiment follows a 2 algorithm (FOCL with greedy literal deletion vs. FOCL with optimal literal deletion) x 4 modification (1, 3, 5, and 7 modifications) design. The only modification to the domain theory in this experiment was randomly adding one or more literals to a clause (using the algorithm described under adding clauses in Section 4.1). The dependent variables measured on 20 trials were the accuracy of the two learning algorithms and the number of times the information gain was computed during deletion of literals. In addition, the outputs of the two simplification algorithms were compared and the number of times they produced identical results was recorded.

An analysis of variance indicates that there is a significant difference in the amount of work performed by the two deletion algorithms. $F(1,152) = 9.03$ ($p < .005$). However, the algorithm did not have a significant effect on the accuracy of the hypothesis ($F(1,152) < 1.0$). During these runs, the more expensive optimal algorithm made 126 deletions during the operationalization process. Of these, the greedy algorithm performed the same deletion 124 times.

5 Related Work

In this section, we compare FOCL to several systems that combine explanation-based and empirical learning.

5.1 A-EBL

The A-EBL system (Cohen, 1990) is designed to handle overly general domain theories. It operates by finding all proofs of all positive examples, and using a greedy set covering algorithm to find a set of operational definitions that cover all positive examples and no negative examples. A similar set covering behavior occurs in FOCL when it deals with overly general domain theories which arise from superfluous clauses (see Figure 2). However, FOCL does not need 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 missing preconditions in clauses (i.e., a missing literal), and overly general domain theories caused by additional clauses

5.2 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 lie in the search control strategies they employ. FOCL uses a hill-climbing approach 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. However the cost of running a best-first algorithm is very high, since the branching factor is a function of the number of variabilizations of all predicates (Pazzani & Kibler, 1990) and the best-first search algorithm requires saving all previous states.

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 all extensions. The heuristics in ML-SMART have not been subject to systematic experimentation of the type we performed in Section 4. As a consequence, it is unclear how well they deal with various types of incomplete and incorrect domain theories and whether they tolerate noise in the training data.

5.3 EITHER

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

In this paper we have described a concept learner, FOCL, that integrates empirical and explanation-based learning in a uniform manner. We have focused on how the learning algorithms are integrated and ignored features of FOCL such as typing of variables, pruning search spaces, and learning numeric thresholds. As a result of the integration of learning methods and a uniform evaluation function applied to each method, FOCL advantageously uses both incorrect and incomplete theories and tolerates classification noise in the training data.

Acknowledgements

We would like to thank Ross Quinlan for his advise on FOIL, Dennis Kibler for helping with a complexity analysis of FOIL, Kamal Ali for helping with the implementation, and Tim Cain and Caroline Ehrlich for commenting on an earlier draft of this paper.

References

Ali, K. (1989). Augmenting domain theory for explanation-based generalization. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 40–42). Ithaca, NY: Morgan Kaufmann.

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.

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

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

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

Fawcett, T. (1989). Learning from plausible explanations. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 37–39). Ithaca, NY: Morgan Kaufmann.

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

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.

Keller, R. (1987). Defining operationality for explanation-based learning. *Proceedings of the Sixth National Conference on Artificial Intelligence* (pp. 482–487). Seattle, WA: Morgan Kaufmann.

Lebowitz, M. (1986). Integrated learning: Controlling explanation. *Cognitive Science, 10,* 219–240.

Michalski, R., & Ko, H. (1988). On the nature of explanation or why did the wine bottle shatter? *Proceedings of the AAAI Symposium on Explanation-based Learning* (pp. 12–16). Stanford, CA: Morgan Kaufmann.

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.

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

Pazzani, M., & Kibler, D. (1990). *The utility of knowledge in inductive learning* (Technical Report No. 90-18). Irvine: University of California, Department of Information & Computer Science.

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.

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

Rajamoney, S., & DeJong, G. (1987). The classification, detection and handling of imperfect theory problems. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* (pp. 205–207). Milan: Italy: 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.

VanLehn, K. (1987). Learning one subprocedure per lesson. *Artificial Intelligence, 31,* 1–40.

Wilkins, D., & Tan, K. (1989). Knowledge base refinement as improving an incorrect, inconsistent and incomplete domain theory. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 332–337). Ithaca, NY: Morgan Kaufmann.