Detecting and Correcting Errors in Rule-Based Expert Systems: An Integration of Empirical and Explanation-based Learning

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

Technical Report 90–38

November 16, 1990

Appeared in *Proceedings of the 5th AAAI Knowledge Acquisition for Knowledge-based Systems Workshop.*

This research is supported by a National Science Foundation Grant IRI-8908260 and by the University of California, Irvine through an allocation of computer time. We would like to thank Ross Quinlan for his advice on FOIL, and Caroline Ehrlich for her comments on an earlier draft of this paper.

Detecting and Correcting Errors in Rule-Based Expert Systems: An Integration of Empirical and Explanation-based Learning

Michael J. Pazzani and Clifford A. Brunk
Department of Information and Computer Science
University of California,
Irvine, CA 92717
714-856-5888
(pazzani@ics.uci.edu)
(brunk@ics.uci.edu)

Abstract

In this paper, we argue that techniques proposed for combining empirical and explanation-based learning methods can also be used to detect errors in rule-based expert systems, to isolate the blame for these errors to a small number of rules and suggest revisions to the rules to eliminate these errors. We demonstrate that FOCL, an extension to Quinlan's FOIL program, can learn in spite of an incorrect domain theory (e.g., a knowledge base of an expert system that contains some erroneous rules). A prototype knowledge acquisition tool, KR-FOCL, has been constructed that can utilize a trace of FOCL to suggest revisions to a rule base.

Introduction

Buchanan et al. (1983) provide an overview of the expert system development process. After identifying problem characteristics, designing representations for the available data and the system's hypotheses, a prototype is constructed. Next, a cycle is repeated in which the prototype is expanded to cover more problems, tested and revised as necessary to improve accuracy on known problems. Occasionally during this testing phase, it is necessary to reconsider the choices made for representation or to clarify the problem being addressed.

Knowledge acquisition tools excel at the creation of a prototype. For example, ETS (Boose, 1984) automates the processes of eliciting problem characteristics, identifying important domain relations, and generating initial rules for a prototype. However, once the initial prototype is constructed, the rules must be revised manually. Similarly, KNACK (Klinker et al., 1989) and SALT (Marcus, McDermott & Wang, 1985) assist in interviewing domain experts to elicit the knowledge required to instantiate a rule-based expert system, but do not fully address the revision of the rule base when errors are discovered.

TEIRESIAS (Davis, 1978) supports the expert and knowledge engineer in revising a rule base. It provides tools to explain the inference path taken by an expert system. If an expert believes this line of reasoning to be erroneous, the rules involved can also be modified by use of TEIRESIAS. This knowledge refinement tool can assist the user in modifying a rule by use of general rule models that indicate what type of information a rule should be encoding. However, much of the effort is left to the expert and knowledge engineer to assign blame for an error to a particular rule, to identify what particular revisions are necessary and to evaluate the impact of making these revisions.

We are interested in providing a tool to help automate the process of revising a prototype when an error is encountered. We restrict our attention to backward-chaining rule-based expert systems that perform classification tasks (Clancey, 1984). The tool, KR-FOCL partially automates the task of identifying rules responsible for errors in expert systems. KR-FOCL is based on a machine learning program, FOCL, developed at the University of California, Irvine (Pazzani & Kibler, 1990). FOCL was designed to learn constant-free Horn-Clause concepts by combining explanation-based and empirical learning. As a consequence of this combination of learning methods, FOCL can tolerate incomplete and incorrect domain theories (Mitchell, Keller, & Kedar-Cabelli, 1986; Rajamoney, & DeJong, 1987). FOCL uses the rule base of an expert system as the domain theory required by explanation-based learning and processes a collection of examples. If the rule base correctly classifies every example, then FOCL creates an operational definition of the classifications performed by the expert system by using only explanation-based learning techniques.[1] On the other hand, if the rule base incorrectly classifies some examples, FOCL will use some combination of explanation-based and empirical learning techniques. In this paper, we show that the conditions learned by the empirical techniques and a description of when the empirical techniques are needed provides

1. Note that we are not advocating replacing the rule base of the expert system with the operational definition of the concept created by explanation-based learning. This operational definition could be used to classify new examples. However, since it removes all of the intermediate hypotheses, the operational concept will be of little use in explaining how the expert system solved a particular problem (Swartout, 1981).

information that focuses the assignment of blame for an error to a particular rule and the revision of erroneous rules.

Table 1 contains a trace of an interaction with KR-FOCL. In this example, FOCL was given a rule-base of a small expert system designed to indicate when a student is required to pay back an educational loan. This expert system was constructed by an undergraduate student at the University California, Irvine who was employed as a loan processor. Four errors were intentionally introduced into the rule base. Appendix I contains a listing of the expert system rule base. FOCL is also given a collection of 48 examples. The examples were evenly divided between those students who were required to make a loan payment and those who were not. With the errors introduced into the rule-base, 21 of the 24 positive examples and 11 of the 24 negative examples are processed correctly by the expert system. The revisions to the rule-base will focus on changing the rules responsible for misclassifying these examples.

The expert system uses a LISP representation for rules. The first element of a list is a predicate name. The remaining elements are arguments to the predicates. Variables are preceded by question marks. For example, the term:[2]

```
    (AND (filed_for_bankruptcy ?s) (disabled ?s))
```

can be read as "The student has filed for bankruptcy and the student is disabled."

Note that KR-FOCL is intended to be used by someone familiar with the domain. This person must approve the changes suggested and help with assignment of blame after KR-FOCL has localized blame to a small number of rules.

In the remainder of this paper, we first describe the learning system, FOCL. Next, we discuss the class of errors in a rule base that FOCL can tolerate. Finally, we report on how KR-FOCL can be used to revise a knowledge-base.

Background: FOIL and FOCL

In order to learn the complex concepts encoded in the knowledge-base of an expert system, it is necessary to have a learning program capable of acquiring relational concepts. Recently, Quinlan (1990) has introduced a learning system called FOIL that is capable of learning Horn Clause concepts such as those used by backward-chaining rule interpreters. FOIL is a purely inductive system. We have produced an extension to FOIL, called FOCL, that combines the inductive component of FOIL with an explanation-based learning component.

2. Note that we have edited FOCL's notation slightly to improve readability. In particular, FOCL uses the position of elements in a rule to indicate the antecedent, the consequent and the presence of conjunction. That is, instead of:

```
    (between ?x ?y) IF (AND (less-than ?x ?z)(less-than ?z ?y))
```

FOCL uses:

```
    ((between ?x ?y ?z) (less-than ?x ?y)(less-than ?y ?z)).
```

In addition, since FOCL uses constant-free literals, so that

```
    (less-than ?x 5)
```

must be represented as:

```
    (AND (less-than ?x ?y)(five ?y)).
```

Table 1. A trace of KR-FOCL

```
1a. Should condition (never_left_school ?s) be added to clause 0 of
enrolled_in_more_than_five_units:
(AND (enrolled ?s ?school ?units) (school ?school) (> ?units 5))
> no

1b. Should conditions (never_left_school ?s) be added to clause 0 of
continuously_enrolled:
(enrolled_in_more_than_five_units ?s)
> yes

2. Should clause 0 of eligible_for_disability_deferment:
(AND (filled_for_bankruptcy ?s) (disabled ?s))
 be replaced with (disabled ?s)
> yes

3a. Should condition (unemployed ?v1) be added as a new clause for
no_payment_due
> no

3b. should condition (unemployed ?v1) be added as a new clause for
eligible_for_disability_deferment
> no

3c. Should condition (unemployed ?v1) be added as a new clause for
eligible_for_financial_deferment
> yes

4. Clause 4 of eligible_for_deferment:
(eligible_for_disability_deferment ?s)
 was unused.  note: (eligible_for_disability_deferment ?s) has been revised
 should it be deleted?
> no

5. Clause 1 of eligible_for_financial_deferment:
(AND (enrolled ?s ?c ?u) (uci ?c))
 was unused.  Should it be deleted?
> yes
```

FOIL

FOIL (Quinlan, 1990) inductively generates constant-free Horn-clause theories in a manner similar to that used by ID3 (Quinlan, 1986) to generate decision trees with attribute-value tests. In particular, FOIL uses a divide-and-conquer approach guided by a heuristic based on information theory. In order to review Quinlan's approach, we need to introduce some terminology. A Horn-clause definition for a concept $(P_0 \ldots ?V_{0,1} ?V_{0,n_0})$ consists of a disjunctive set of clauses. Each clause consists of a head and a conjunction of literals:

$(P_0\ ?V_{0,1}\ \ldots\ ?V_{0,n_0})$ if (AND $(P_1\ ?V_{1,1}\ \ldots ?V_{1,n_1}) \ldots (P_m\ ?V_{m,1}\ \ldots ?V_{m,n_m})$).

We will call $(P_m\ ?V_{m,1}\ \ldots ?V_{m,n_m})$ a variabilization of a predicate P_m. Since FOIL is currently under development, we present here only its essential characteristics. FOIL is given a collection of labeled examples, a set of known predicates, and an unknown predicate P_0 of known arity that FOIL is to learn. The task is to determine a Horn-clause definition of P_0 in terms of the given predicates as well as (in a limited manner) the predicate P_0.

In our description of FOIL, Pos is the set of positive instances and Neg is the set of negative instances. Similar to AQ (Michalski, 1980), FOIL has two main phases: adding clauses to the theory until every positive instance is covered and forming clauses that do not contain any negative instances. Positive instances are covered as follows:

```
Until Pos is empty
      Construct a clause that covers some positive instances
            and avoids all negative instances.
      Add  clause to the theory.
      Remove those elements of Pos that are covered by the new clause.
```

Constructing a clause that misses all negative instances can also be described simply. Note that the clause $(P_0\ ?V_{0,1}\ \ldots\ ?V_{0,n_0})$ if true covers all positive instances. To avoid negative examples, this clause is specialized in the following manner:

```
Let   (P_0 ?V_{0,1} ... ?V_{0,n_0}) if true be the initial clause
Let Pos be the positive examples not satisfied by the current definition
Let Neg be the negative examples
Until Neg is empty
      Choose the predicate variabilization with the maximum information
gain.
      If the maximum information gain ≤ 0, then exit with failure
      Conjoin the  predicate variabilization the body of the clause.
      Let Pos be all extensions of Pos that are satisfied by the term.
      Let Neg be all extensions of Neg that are satisfied by the term.
```

At this 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, FOIL measures the information gain, which is defined differently from ID3. Without going into the exact computation, (see Quinlan, 1990) we note that, in effect, the information metric determines the number of positive and negative examples satisfied and those predicate variabilizations that correctly classify more examples have the higher information gain.

FOCL

Our work on FOCL is motivated by three factors:

1. FOIL is a purely inductive learning system and cannot make use of any available domain knowledge. When there is domain knowledge available, analytic learning

systems can learn from fewer examples than inductive learning systems because the domain knowledge can be used to constrain the hypothesis space.

2. The hill-climbing control structure of FOIL can prevent it from learning some concepts. For example, both `(less-than ?a ?b)` and `(less-than ?b ?c)` may have negative information gain, while the conjunction of these two terms may have positive information gain. As a consequence, FOIL is not able to learn the conjunction of these two literals.

3. The size of the hypothesis space searched by FOIL can be quite large (see Pazzani & Kibler, 1990 for an analysis of the complexity of FOIL). FOIL computes the information gain of all orderings of all the bound variables and new variables for all known predicates. By using domain knowledge, it may be possible to prune the number of predicate variabilizations considered.

FOCL, like FOIL, uses an information-based metric to evaluate when to add a literal to a clause under construction. However, unlike FOIL literals may be proposed either by an inductive component or by an explanation-based component. The result of this integration is that the system can make use of domain knowledge when available to constrain the search for a concept definition. However, when the domain knowledge is incomplete or incorrect, the literals proposed by the explanation-based component will have less information gain than the literals proposed by the inductive component. As a consequence, FOCL utilizes incomplete and incorrect domain theories and can make up for these deficiencies by relying on an inductive component to complete the concept definition.

Like all explanation-based learning systems, FOCL requires a target concept, (i.e., a non-operational definition of the concept to be acquired), a domain theory theory (i.e., a set of rules, such as the rule base of the expert system, that relates the non-operational definition to operational predicates), a set of operational predicates (i.e., features that the examples are expressed in terms of), and a set of classified training examples.[3] Table 2 displays this information for the student loan example and a positive and a negative training example. Note that the domain theory in Appendix I incorrectly classifies these two examples. S3 is erroneously classified as a negative example (because the rule for disability deferment has been modified by adding an extra condition) and n17 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).

FOCL differs from FOIL only in how FOCL selects literals to add to a clause. FOCL first computes the information gain of each literal in the target concept. In Table 2, the target concept is a single literal. In general, the target concept can be a conjunction of literals of the form:

$$\texttt{(AND (}P_1\ ?V_{1,1}\ \ldots ?V_{1,n_1}\texttt{)}\ldots\texttt{(}P_m\ ?V_{m,1}\ \ldots ?V_{m,n_m}\texttt{))}.$$

If any literal has positive information gain, then FOCL operationalizes that literal. Otherwise, FOCL uses the inductive method of FOIL and computes the information gain of all

3. Note that FOIL only requires a set of operational predicates and a set of training examples.

Table 2. Input to FOCL

Target concept:	(no_payment_due ?s)
Domain theory:	{see Appendix I}
Operational Pred:	{ (enrolled ?student ?school ?units) (longest_absence_from_school ?student ?months) (enlist ?student ?organization) (unemployed ?student) filled_for_bankruptcy (filled_for_bankruptcy ?student) (disabled ?student) (school ?organization) (armed_forces ?organization) (peace_corps ?organization) }
Positive Examples:	{s1,s2...,s24}
Negative Examples:	{n1,n2...,n24}

A Positive Example	**A Negative Example**
(longest_absence_from_school s3 3)	(longest_absence_from_school n17 12)
(enrolled s3 OCC 5)	(enrolled n17 UCI 10)
(disabled s3)	

variabilizations of all predicates (both operational and nonoperational) and selects the predicate variabilization with the maximum information gain. This strategy insures that FOCL uses as much of the user provided domain theory as possible, but disregards those parts of the domain theory that misclassify large numbers of examples.

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 a single positive example. As in EBL, the operational definition for a predicate may specialize the predicate if the domain theory is disjunctive (i.e., if there are multiple clauses for any non-operational predicate). In EBL, the predicates that are the leaves of the proof tree of the single training example are used as the operational definition. In FOCL, the information gain metric is used to determine how to expand a proof tree, in the following manner:

```
operationalize(Term, Pos, Neg):
let Body be the empty set.
for each Clause in the definition of Term
   compute_gain(Clause, Pos, Neg)
for the Clause with the maximum gain
   for each term T in Clause
        if T is operational
        then add T to Body
        else add operationalize(T, Pos, Neg) to Body
```

A final difference between FOCL and FOIL is that the inductive component of FOCL can make use of nonoperational predicates. The inductive component of FOCL computes the information gain of all possible variabilizations of the nonoperational predicates in the same manner as the operational predicates. If a nonoperational predicate has the maximum information gain, it is operationalized in the same manner that the target concept is operationalized by the explanation-based learning component of FOCL.

An example will help to illustrate how FOCL operates. Later, we will use this same example to show how KR-FOCL can be used to revise an expert system. The trace in Table 3 shows FOCL operating with the incorrect domain theory from Appendix I. In the first step, FOCL tries to operationalize the concept `no_payment_due`. There are two clauses that can be used to prove that no payment is due and FOCL computes the information gain of both and selects the alternative with the highest information gain: `eligible_for_deferment`. There are five alternative ways of proving this and `eligible_for_military_deferment` has the highest gain. Finally, this rule has only one alternative. Since this alternative is operational, it is added as a new literal. The new literal does not cover any negative examples, so it is a new clause. In Steps 2-4, this process is repeated and rules corresponding to `eligible_for_financial_deferment`, `eligible_for_student_deferment`, `eligible_for_peace_corps_deferment` are operationalized.

In Step 5, the predicate `continuously_enrolled` is operationalized. However, the literal formed by operationalizing this predicate, also covers thirteen negative examples (i.e., the expert system would erroneously report that these students should not repay their loans, but they should). At this stage, FOCL tries to induce another literal that excludes these negative examples, and satisfies some of the positive examples that are covered by the operationalized literal for `continuously_enrolled`. The literal, `never_left_school`, has the highest information gain. Because this predicate is not operational (i.e., it is defined by a rule rather than a set of facts that are true of students), this predicate is also operationalized. The resulting literal is conjoined with the operationalization of `continuously_enrolled` and a new clause is formed that does not cover any remaining examples. Note that the first part of this clause was formed by EBL (operationalizing a target concept with positive information gain) and the second part was formed inductively (by searching all variabilization of all operational and all nonoperational predicates).

At this point, the target concept no longer has positive information gain. This occurs because the set of positive examples is reduced by eliminating those examples that are satisfied by each new clause created. Since the target concept no longer will correctly classify the remaining examples, only inductive techniques are utilized. In Step 6, it is induced that disabled persons are not required to pay back loans and in Step 7, it is determined 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 process terminates.

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

Table 3. A trace of FOCL

```
1. >(continuously_enrolled ?v1)                                  gain    1.4
>(eligible_for_deferment ?v1)                                    gain   14.0
>operationalizing (eligible_for_deferment ?v1)
>>(eligible_for_military_deferment ?v1)                          gain    6.0
...
>>(eligible_for_disability_deferment ?v1)                        gain    1.0
>>operationalizing (eligible_for_military_deferment ?v1)
>>>(AND (enlist ?v1 ?v101) (armed_forces ?v101))                 gain    6.0

New Literal: (AND (enlist ?v1 ?v101) (armed_forces ?v101))
New Negatives: {}

New Clause: (AND (enlist ?v1 ?v101) (armed_forces ?v101))

2. New Clause:  (filled_for_bankruptcy ?v1)
3. New Clause:  (AND (enrolled ?v1 ?v103 ?v102)(school ?v103)(> ?v102 11))
4. New Clause:  (AND (enlist ?v1 ?v101) (peace_corps ?v101))
5. >(continuously_enrolled ?v1)                                  gain    1.8
>(eligible_for_deferment ?v1)                                    gain    0.0
>operationalizing (continuously_enrolled ?v1)
>(enrolled_in_more_than_five_units ?v1)                          gain    1.8
>operationalizing (enrolled_in_more_than_five_units ?v1)
(and (enrolled ?v1 ?v103 ?v102) (school ?v103) (> ?v102 5))      gain    1.8
New Literal: (AND (enrolled ?v1 ?v103 ?v102) (school ?v103) (> ?v102 5))
New Negatives: {n23 n22 n20 n18 n17 n16 n15 n14 n13 n12 n11 n10 n9}

>(enrolled ?v1 ?v104 ?v105)                                      gain    0.0
>(enlist ?v1 ?v106)                                              gain    0.0
>(disabled ?v1)                                                  gain    4.5
>(filled_for_bankruptcy ?v1)                                     gain    0.0
...
>(never_left_school ?v1)                                         gain    6.1
>operationalizing (never_left_school ?v1)
>(AND (longest_absence_from_school ?v1 ?v104)(> 6 ?v104))        gain    6.1
New Literal: (AND (longest_absence_from_school ?v1 ?v104)(> 6 ?v104))
New Negatives: {}
New Clause: (AND (enrolled ?v1 ?v103 ?v102) (school ?v103)(> ?v102 5)
                 (longest_absence_from_school ?v1 ?v104)(> 6 ?v104))

6. >(continuously_enrolled ?v1)                                  gain   -2.7
>(eligible_for_deferment ?v1)                                    gain    0.0
...
>(disabled ?v1)                                                  gain    9.2
>(unemployed ?v1)                                                gain    4.6
...
New Literal: (disabled ?v1)
New Negatives: {}
New Clause: (disabled ?v1)
7. New Clause: (unemployed ?v1)
```

Errors in rule bases

KR-FOCL is designed to deal with four types of errors in a rule base:

1. A clause has an extra literal (or literals) added. This extra literal makes the rule overly specific so that some positive examples will be classified incorrectly.

2. A clause has a literal (or literals) missing. This missing literal makes the rule overly general so that some negative examples will be classified incorrectly.

3. A rule has an extra clause (or clauses) added. This extra clause makes the rule overly general.

4. A rule has an clause (or clauses) missing. This missing clause makes the rule overly specific.

The rule base displayed in Appendix I has one of each type of these errors. Note that combinations of these errors can account for other erroneous rules. For example, using the wrong literal (e.g., `(> ?x 7)` instead of `(< ?x 7)`), can be viewed as having a missing literal and an extra literal.

KR-FOCL: A Knowledge Revision Tool

We are advocating that part of the documentation for an expert system consists of a set of correctly classified examples that exercise every clause of every rule of the expert system. For example, for a fault diagnosis system, the examples might consist of mappings between sensor readings and failures; for a medical expert system, the examples might consist of mappings between patient symptoms and diseases (or treatments). These examples are probably best collected incrementally, as each clause is entered.

FOCL is capable of using these examples to learn an operational concept definition in spite of errors in the rule base. However, FOCL does not fix the rule base to correct these errors. KR-FOCL is designed to use a trace of how FOCL learns to suggest revisions to the knowledge base. The following information is retained for each clause learned:

1. The clauses (if any) of each rule that were operationalized by the explanation-based learning component. We will call these clauses *operationalized clauses*.

2. The variabilized predicates (if any) that were added by the induction component. We will call these predicates *induced predicates*.

As FOCL runs, it records the operationalized clauses and the induced predicates for each clause it creates. We will call those clauses of the domain theory that were never operationalized, *unoperationalized clauses*. Table 4 summarizes this information for the example in Table 3.

KR-FOCL analyzes the trace information to suggest revisions to the knowledge base. There are three alternatives that can occur for each clause that FOCL learns:

Table 4. Trace information used by KR-FOCL

Clause	operationalized clauses		induced predicates
	#	predicate	
1.	1	no_payment_due	{}
	0	eligible_for_deferment	
	0	eligible_for_military_deferment	
2.	1	no_payment_due	{}
	2	eligible_for_deferment	
	0	eligible_for_financial_deferment	
3.	1	no_payment_due	{}
	3	eligible_for_deferment	
	0	eligible_for_student_deferment	
	0	enrolled_in_more_than_eleven_units	
4.	1	no_payment_due	{}
	1	eligible_for_deferment	
	0	eligible_for_peace_corp_deferment	
5.	0	no_payment_due	(never_left_school ?v1)
	0	continuously_enrolled	
	0	enrolled_in_more_than_five_units	
6.		{}	(disabled ?v1)
7.		{}	(unemployed ?v1)

unoperationalized clauses:

4	eligible_for_deferment
1	eligible_for_financial_deferment
0	eligible_for_disability_deferment

1. The set of operationalized clauses is non-empty and the set of induced predicates is empty. This is a sign that none of the operationalized clauses used need to be modified to correctly classify the training examples.

2. The set of operationalized clauses is non-empty and the set of induced predicates is non-empty. This may be a sign that one or more of the operationalized clauses need to be modified by adding an extra condition from the induced predicates. The user is prompted to verify one of these possibilities.

3. The set of operationalized clauses is empty and the set of induced predicates is non-empty. This may be a sign that one of the unoperationalized clauses needs to be changed. FOCL searches for clauses in the rule base that meet one of two conditions:

a. The induced predicates subsume an unoperationalized clause. This may be a sign that the clause can be replaced by the induced predicates. In effect, this modification removes superfluous literals from a clause.

b. The induced predicates are subsumed by an unoperationalized clause. This also can be a sign that the clause can be replaced by the induced predicates. In effect, this modification adds extra literals to a clause.

Finally, this condition may also be a sign that the induced predicates may be another clause for the top level predicate or for a predicate that is part of an unoperationalized clause. Note that the induced predicates in FOCL need not be operational, so that the induced predicates can use the existing vocabulary of the knowledge base.

KR-FOCL asks the user to confirm any changes that are suggested. Finally, KR-FOCL suggests deleting unoperationalized clauses. The trace in Table 1 illustrates the types of revisions that KR-FOCL suggests.

The first example in Table 1 is a case in which the operationalized clauses are non-empty and the set of induced predicates are non-empty. This occurred when FOCL was learning the fifth clause (see Table 3). In this clause, FOCL operationalized clauses for `continuously_enrolled` and `enrolled_in_more_than_five_units`. In this situation, FOCL also had to induce a condition, `never_left_school`, to rule out additional negative examples. KR-FOCL asks the user to determine if either of the operationalized clauses should be revised by adding the induced predicate. Note that modifying either clause will result in correct classification of this training set. The choice made by the user will best preserve the structure of the rule base that supports understandable explanations. That is, it makes more sense to say that a student is continuously enrolled if the student has never left school and the student is enrolled in five or more units, than it does to say that a student is enrolled in five or more units if the student has never left school and the number of units in which the student is enrolled is greater than 5.

The second example in Table 1 exemplifies the case when the set of operationalized clauses returned by FOCL is empty (see clause 6 of Table 3). KR-FOCL finds an unoperationalized clause, `(AND (filed_for_bankruptcy ?s) (disabled ?s))`, that is subsumed by the induced predicate, `(disabled ?s)`, and asks the user if the clause can be replaced.

The third example in Table 1 is similar to the second example. However, this time there is no clause that is subsumed by the induced predicate `(unemployed ?v1)` (see clause 7 in Table 3). In this case, KR-FOCL asks if the induced predicate can be a new clause for the top level clause, `no_payment_due`. Although this would result in correct classifications, the user responds negatively. Next, KR-FOCL asks if the induced predicates should be a new clause of any rule for a literal in a clause that was not operationalized. The rationale here is that some clause may not have been operationalized not because the clause itself needs to be modified, but because a rule used to establish one of the conditions of the clause may have another alternative. In this case, clause 4 of `eligible_for_deferment` is unused and KR-FOCL next asks if there should be a new clause for `eligible_for_disability_deferment`. The user responds negatively. Although this would result in correct classifications, it would not make much sense to say that a student is eligible for a disability deferment if the student is unemployed. Finally, KR-FOCL asks the user about adding an extra clause to any rule that has an unoperationalized clause. The user confirms that `(unemployed ?v1)` is another alternative for `eligible_for_financial_deferment`.

Finally, KR-FOCL asks the user to confirm the deletion of any unoperationalized clauses and warns the user about clauses that use predicates whose definition has changed. The user decides to retain the clause that states a student is eligible for a deferment if the student is eligible for a disability deferment (example 4 in Table 1) and decides to delete the clause that states that a student is eligible for a financial deferment if the student is enrolled at UCI.

Limitations

The current implementation of KR-FOCL has several limitations. First, KR-FOCL does not explain why it recommends a certain revision. We are planning an extension that would address this problem by keeping track of the examples that motivate the recommendation. In its current form, we are assuming that a knowledgeable expert either forgot to mention a condition, or mentioned a superfluous condition. When KR-FOCL suggests a revision, it is expected that the expert will realize that the revision will lead to improved performance. In the future, we hope that by providing an expert with a summary of the examples in which the rules in a knowledge base lead to errors, and a revision which eliminates these errors, KR-FOCL may be used by an expert to help discover the correct rules.

Currently, we have only tested KR-FOCL on small examples with only a few errors. As the size of the rule base and the number of errors increases, it may be necessary to find a way to organize the possible revisions to the knowledge base. For example, instead of serially asking verification questions about a large number of possibilities, it may be better to present the user with a menu of these alternatives. In addition, it might be best to order the revisions to the knowledge base by initially focusing on those that will most affect the accuracy of the expert system. Finally, the capabilities of a program such as SEEK (Politakis & Weiss, 1984) to quickly access the impact of a revision would be useful. Currently, the user must run KR-FOCL again to determine the impact of a change.

When the operationalized clauses and induced predicates are non-empty, KR-FOCL suggests revising every operationalized clause by adding the induced predicates. This typically focuses the revisions to a small portion of the knowledge base. We could further supervise this process by making use of rule models (Davis, 1978) that provide information concerning the classes of predicates that are usually used in clauses that propose certain classes of hypotheses.

Finally, KR-FOCL is limited by the underlying machine learning technology.[4] It cannot operate if FOCL cannot solve the problem by using hill-climbing search, or if the problem cannot easily be expressed by constant-free Horn-Clauses. It cannot operate with rules that require certainty factors, rules that deal with large numbers of constants or real-valued constants[5]. In addition, it cannot operate if the expert system relies on a complex conflict resolution strategy to determine the applicability of rules.

Conclusions

In this paper, we have focused on how machine learning technology, in particular programs that integrate empirical and explanation-based learning, can be used to revise the rule bases of expert systems. It is worth noting that knowledge acquisition technology can also address some problems in machine learning. For example, ETS (Boose, 1984) can help automate the process of selecting a vocabulary for training examples. Many successful inductive programs (e.g., Michalski & Chilausky, 1980; Michie, 1983; Quinlan, 1986) require an expert to determine a set of potentially relevant features. In addition, information such as the rule models used by TEIRESIAS may be useful in constraining the hypotheses considered by a learning program.

Acknowledgements

We'd like to thank Ross Quinlan for his advice on FOIL and Caroline Ehrlich for her comments on an earlier draft of this paper. This research is supported by a National Science Foundation Grant IRI-8908260 and by the University of California, Irvine through an allocation of computer time.

4. Note that KR-FOCL gets around a major limitation of SEEK2 (Ginsberg, 1988), in that KR-FOCL can make use of domain theories expressed as Horn-clauses instead of purely propositional domain theories.

5. Note that if there are few constants, it is practical to add predicates that are true of these constants. For example `(color ? x red)` can be replaced with `(AND (color ?x ?y) (red ?y))`. This is not practical with real numbers.

Bibliography

Boose, J. (1984). Personal construct theory and the transfer of human expertise. *Proceedings of the National Conference on Artificial Intelligence*. Austin, TX: Morgan Kaufmann.

Buchanan, B., Barstow, D., Bechtal, R., Bennet, J., Clancey, W., Kulikowski, C., Mitchell, T., & Waterman, D. (1983). Constructing an expert system. In Hayes-Roth, F., Waterman, D., and Lenat, B., *Building Expert Systems*. London: Addison-Wesley Publishing Company.

Clancey, W. (1984). Classification problem solving. *Proceedings of the National Conference on Artificial Intelligence*. Austin, TX: Morgan Kaufmann.

Davis, R. T. (1978). Model-directed learning of production rules. In D. Waterman & F. Hayes-Roth (Eds.), *Pattern-directed inference systems*. New York: Academic Press.

Klinker, G., Boyd, C., Genetetr, S., & McDermott, J. (1987). A KNACK for knowledge acquisition *Proceedings of the Sixth National Conference on Artificial Intelligence*. Seattle, WA: Morgan Kaufmann.

Ginsberg, A. (1988). *Automatic refinement of expert system knowledge bases*. London: Pittman.

Marcus, S. McDermott, J. & Wang, T. (1985). Knowledge acquisition for constructive systems. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*. Los Angeles, CA: Morgan Kaufmann.

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

Michalski, R. & Chilausky, R. (1980) Learning by being told and learning from examples: An experimental comparison of the two methods of knowledge acquisition. *International Journal of Policy Analysis and Information Systems, 4*, 125-161.

Michie, D. (1983). Inductive rule generation in the context of the fifth generation. *Proceedings of the International Machine Learning Workshop. University of Illinois, 65-70.*

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

Pazzani & Kibler (1990). The utility of knowledge in inductive learning. Technical Report. University of California, Irvine.

Politakis, P & Weiss, S. (1984). Using empirical analysis to refine expert system knowledge Bases. *Artificial Intelligence, 22*, 23-48.

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

Quinlan, J. R. (1990). *Learning logical definitions from relations*. Technical Report, University of Sydney.

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. 205-210). Milan, Italy: Morgan Kaufmann.

Swartout, W.. (1981). Explaining and justifying in expert consulting programs. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence* (pp. 203-208). Vancouver, BC: Morgan Kaufmann.

Appendix I. Rule base of the loan repayment expert system.

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) IF (continuously_enrolled ?s)
(no_payment_due ?s) IF (eligible_for_deferment ?s)

(continuously_enrolled  ?s) IF (AND ~~(never_left_school ?s)~~
                                    (enrolled_in_more_than_five_units ?s))

(eligible_for_deferment ?s) IF (eligible_for_military_deferment ?s)
(eligible_for_deferment ?s) IF (eligible_for_peace_corps_deferment ?s)
(eligible_for_deferment ?s) IF (eligible_for_financial_deferment ?s)
(eligible_for_deferment ?s) IF (eligible_for_student_deferment ?s)
(eligible_for_deferment ?s) IF (eligible_for_disability_deferment ?s)

(eligible_for_military_deferment ?s) IF (AND (enlist ?s ?a)
                                             (armed_forces ?a))

(eligible_for_peace_corps_deferment ?s) IF (AND (enlist ?s ?a)
                                                (peace_corps ?a))

(eligible_for_financial_deferment ?s) IF (filed_for_bankruptcy ?s)
~~(eligible_for_financial_deferment ?s) IF (unemployed ?s)~~
**(eligible_for_financial_deferment) IF (AND (enrolled ?s ?c ?u)
                                             (uci ?c)))**

(eligible_for_student_deferment ?s)IF(enrolled_in_more_than_eleven_units ?s)

(eligible_for_disability_deferment ?s) IF **(AND(filed_for_bankruptcy ?s)**
                                           (disabled ?s))

(never_left_school ?s) IF (AND (longest_absence_from_school ?s ?a) (> 6 ?a))

(enrolled_in_more_than_five_units ?s) IF (AND (enrolled ?s ?school ?units)
                                              (school ?school)
                                              (> ?units 5))

(enrolled_in_more_than_eleven_units ?s) IF (AND (enrolled ?s ?school ?units)
                                                (school ?school)
                                                (> ?units 11))
```