FRST - An Interactive Revision System for Forward Chaining Rule Bases

Ronen Feldman
Math and Computer Science Department
Bar-Ilan University
Ramat-Gan ISRAEL
feldman@bimacs.cs.biu.ac.il

Abstract

One of the main problems in building rule based systems is that the knowledge elicited from experts is not always correct. Therefore there is a need for means of revising the rule base whenever an inaccuracy is discovered. The rule base revision is the problem of how best to go about revising a deficient rule base using information contained in cases that expose inaccuracies. The revision process is very sensitive to implicit and explicit biases that are encoded in the specific revision algorithm employed. In a sense, each revision algorithm must provide two forms of biases. The first bias governs the preferred location in the rule base for the correction, while the second bias governs the type of correction performed. In this paper we present a system for incremental revision of rule bases called FRST (Forward chaining Revision SysTem). This system enables the user to analyze the impact of different revisions and to select the most appropriate revision operator. The user provides the required bias by using a special bias language which was designed specifically for use in rule based systems.

1. Introduction

One of the main problems in building knowledge based systems is that the knowledge elicited from experts is not always correct. In addition, the knowledge revision phase usually takes tremendous amount of time from the expert and the knowledge engineer building the system. Therefore there is a need for means of revising the knowledge base whenever an inaccuracy is discovered . Classical techniques for inductive learning, commonly used in the machine learning community can be of help in automating this tedious task. The theory revision problem in machine learning is the problem of how best to go about revising a deficient knowledge base using information contained in cases that expose inaccuracies. The revision process is very sensitive to implicit and explicit biases that are encoded in the specific revision algorithm employed. In a sense, each revision algorithm must provide two forms of biases. The first bias governs the preferred location in the knowledge base for the correction, while the second bias governs the type of correction performed. In this paper we present a system for writing approximate rule bases which are annotated with explicit bias. The purpose of the revision bias framework is to allow the user to declare its own bias in a simple and structured way, that is to express the conditions on the domain knowledge elicited from the expert, under which a given revision operator should be applied. The knowledge representation language selected in is the rule language of CLIPS, an expert system shell developed at NASA [NASA 1993]. Former approaches to theory revision like EITHER (Ourston and Mooney, 1991), SEEK2 (Ginsberg 1988), KBANN (Towell el al. 1990) and DUCTOR (Cain 1991) concentrated on propositional logic while AUDRY (Wogulis, 1991), FORTE (Richards and Mooney, 1991) and FOCL (Pazzani and Kibler, 1990) concentrated on some variants of pure Prolog. The FRST system presented here is a step forward in the direction of using the experience acquired in current theory revision systems to refinement and revision of complex knowledge bases written using a real expert system. Another system which is capable of revising rule bases is CLIPS-R (Murphy and Pazzani, 1994). The main difference between FRST and CLIPS-R is that CLIPS-R does not give the user a full bias language so that the user can direct the system to the desired revision. The main criteria used by CLIPS-R is syntactic in nature, find the minimal change from the original rule base.

The rest of this paper is organized as follows. In Section 2 we present the problem of incremental revision of rule bases and discuss why is it harder than classical theory revision problem. Section 3 outlines the structure of general forward chaining rule based system and Section 4 describes the basic components of the CLIPS system. In Section 5 we describe our revision algorithm and in Section 6 we describe FRST system which uses this revision algorithm and a special bias language designed to support revision of rule bases. Finally, in Section 7 we present our conclusions and the major contributions of this paper.

2. Incremental Rule Base Revision

Informally, the rule base revision problem is the problem of how best to go about revising a deficient rule base using information contained in cases that expose inaccuracies. In order to characterize those attributes which are desirable in a rule base revision system, let us briefly examine the process by which an expert system is constructed. Typically, a domain expert sits with a knowledge engineer and encodes his or her expert knowledge as a collection of rules. Sometimes more than one domain expert is involved, leading to possible inconsistencies in the rule base, although inconsistencies are not uncommon in large rule bases even if there is only one domain expert. After the knowledge has been encoded, it is used given cases. These cases usually are obtained over a long period of time as the expert system is put to use. As errors and inaccuracies are encountered, the rule base must be refined manually by the knowledge engineer. Some errors may be due to simple encoding errors, while other errors may represent deeper, conceptual, errors in the expert's understanding of the domain.

A rule base revision system should in practice replace this knowledge refinement process. A useful solution to the theory revision problem should integrate smoothly with the traditional approach to knowledge engineering. Thus, it is important that a rule base revision system have certain qualities. First, the system should be *incremental,* meaning it should operate on cases as they are obtained. Rule base revision systems which require all cases be provided in a single batch do not fit well with the traditional knowledge engineering process. Second, a system which reconfigures the rule base so that it no longer makes sense to the knowledge engineer lacks *referential transparency*: a useful system should largely preserve the structure of the original rule base so that, at any time, the knowledge engineer may examine and understand the internal structure of the rule base. Third, a domain expert may know *a priori* which portions of the rule base should be trusted and which portions of the rule base are more conjectural. Thus the changes undertaken by the system should reflect the expert's intuitive *confidence* in the individual components of the rule base. PTR (Feldman 1993, Koppel et al 1994) is a theory revision algorithm that is capable of taking advantage of such explicit biases. PTR can revise propositional theories and relational theories that are written as pure prolog programs. This knowledge representation appears to be too simple and is not used in real world systems. Most expert systems are written using expert systems shells such as G2, CLIPS, RAL and ART-IM. Most of these expert system shells use very similar language for writing the rules that comprise the knowledge base. In this paper we will use the rule language used by CLIPS. Revision of CLIPS rule bases appears to be much more difficult than revision of a relational prolog program since we have a much richer set of revision operators at our disposal and due to the fact that the execution model is forward chaining rather than backward chaining as in prolog programs. A modification of PTR is used in order to locate flawed rule base elements and a special bias language was tailored in order to provide specific bias for selecting the appropriate revision operators to be used with these flawed elements. In the absence of any bias from the user, the system will pick the operator which will make the most conservative revision i.e., the revision which makes the smallest change to the rule base. This appears to be the most reasonable strategy when no explicit guidance is available.

3. Forward Chaining Rule Bases

An expert system consists of 3 main modules: a central control module, a knowledge base and an inference engine. The control module is in charge of the interaction with the user and activation of the inference process based on the description of the current case. The inference engine can operate in one of several inference modes in order to draw conclusions based on description of the case. The rules that are used by the inference engine are stored in the knowledge base. Each rule has two parts the LHS which contain the conditions and the RHS which contain the actions to be performed should this rule be chosen to be fired. There are two major modes of inference used by expert systems, forward chaining and backward chaining. Backward chaining is the mode used when trying to prove a particular conclusion. This mode is also the inference mode used by the prolog inference engine. Forward chaining start from the basic facts and uses the rules in order to reach some conclusion. The major components of a typical expert system are shown in Figure 1.

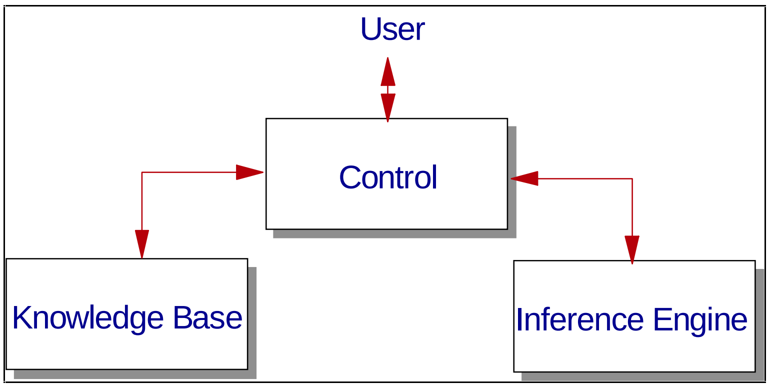


Figure 1 - Expert System Structure

The basic loop in a forward chaining rule base system is:

1. Match: find all the rules whose all conditions are satisfied.
2. Select: apply the conflict resolution strategy to select one of the candidate rules to be fired.
3. Fire: execute the actions of the selected rule.
4. goto 1.

This loop is repeated until there no rule that can be fired. The basic facts known to be true in the world at each stage of the execution are stored in the *working memory*. Each such fact is called *working memory element* (WME). The rules rely on the WMEs for satisfying their conditions (the match step) and then after the chosen rule is fired the actions specified in its RHS are executed and the working memory changed either by the addition of new facts and/or by retraction some old facts.

4. The CLIPS System

CLIPS(C Language Integrated Production System) was developed by at Nasa's Johnson Space Center and is widely used in many locations around the world for developing expert systems applications. The user can define rules, facts, templates and functions. We will concentrate here only on the rules part of a CLIPS program. A typical rule in CLIPS has the following structure:

```
(defrule rule-name
        (declare (salience S))
```

conditions
=>
actions)

The rule will be inserted into the agenda as a candidate for firing if all the conditions are met and then according to the resolution strategy, one rule is selected and all its actions are performed. The rule is assigned a salience S which is a priority to be used by the conflict resolution strategy. The typical actions are: asserting a new fact, retracting a fact, modifying a fact, and printing a message to the standard output. We will concentrate on the actions that are related to facts.

4.1 Example:

The following rules are taken from the auto.clp rule base written by NASA. This is an expert system that diagnoses some simple problems with a car.

```
(defrule unsatisfactory-engine-state-conclusions ""
   (declare (salience 10))
   (working-state engine unsatisfactory)
   =>
   (assert (charge-state battery charged))
   (assert (rotation-state engine rotates)))
```

This rule says that if the engine condition is unsatisfactory that add two assertions: that the battery is charged and that the engine is rotating.

```
(defrule determine-sluggishness ""
   (working-state engine unsatisfactory)
   (not (repair ?))
   =>
   (if (yes-or-no-p "Is the engine sluggish (yes/no)? ")
       then (assert (repair "Clean the fuel line."))))
```

This rule says that if the engine condition is unsatisfactory and no repair was found so far than ask the user if the engine is sluggish and on a positive reply conclude that the need repair is to clean the fuel line.

```
(defrule determine-misfiring ""
   (working-state engine unsatisfactory)
   (not (repair ?))
   =>
   (if (yes-or-no-p "Does the engine misfire (yes/no)? ")
       then
       (assert (repair "Point gap adjustment."))
       (assert (spark-state engine irregular-spark))))
```

This rule says that if the engine condition is unsatisfactory and no repair was found so far than ask the user if the engine misfires and on a positive reply conclude that the need repair is to adjust the point gap and assert that the engine is sparking irregularly.

In the following subsection we will show a trace of the execution of the auto expert system. At each point we can see the rules that are in the conflict set (marked by Activation), the rule which is fired at each stage and the facts which added or retracted from the working memory. In addition the questions presented to the user and the answers given are shown in the appropriate places in the trace.

4.1.1 Sample Execution

```
CLIPS> (run)
FIRE    2 determine-engine-state: ,
Does the engine start (yes/no)? yes
Does the engine run normally (yes/no)? no
==> f-1     (working-state engine unsatisfactory)
==> Activation 0      determine-low-output: f-1,
==> Activation 0      determine-knocking: f-1,
==> Activation 0      determine-misfiring: f-1,
==> Activation 0      determine-sluggishness: f-1,
==> Activation 10     unsatisfactory-engine-state-conclusions: f-1
FIRE    3 unsatisfactory-engine-state-conclusions: f-1
==> f-2     (charge-state battery charged)
==> f-3     (rotation-state engine rotates)
FIRE    4 determine-sluggishness: f-1,
Is the engine sluggish (yes/no)? yes
==> f-4     (repair "Clean the fuel line.")
```

4.2 Erroneous Cases

Each case is annotated with preconditions and postconditions. Preconditions are the facts which constitute the working memory before the expert system is run, and the postconditions are the facts that should constitute the working memory after the expert system is done. The preconditions contain also all answers to the questions so that the system can run in a fully automatic mode. In addition, the preconditions contain also a partial order on rule firings.

A case will be considered as an erroneous if the contents of the working memory after the expert system was done does not match the postconditions of the case. In other words, either there are some missing facts or there some extra facts in the working memory. The other inconsistency that can be recognized is if the order of rule firings does not obey the partial order specified for the case.

4.3 Revision Operators

In this subsection we will list all the revision operators that will be used by the FRST system. The operators are divided into two classes, generalization operators and specialization operators. This classification of the operators is true only if we do not have negation. In the presence of negation then each of the generalization operators might become a specialization operator and vice versa (for more details on the issue of negation see (Koppel et al. 1994)).

4.3.1 Generalization Operators

1. Addition of a new rule.
2. Addition of a new assertion to the rule's RHS.
3. Deletion of rule retraction from the rule's RHS.
4. Deletion of a condition in a rule's LHS.
5. Generalization of a condition by lifting a variable constraint.
6. Generalization of a condition by replacing constants.
7. Generalization of a condition by replacing a relation or template functor by a more general functor.
8. Increasing a rule salience, so that a given assertion will take place.

4.3.2 Specialization Operators

1. Deletion of a rule.
2. Deletion of an assertion from a rule's RHS.
3. Addition of a fact retraction to a rule's LHS.
4. Addition of a condition to a rule's RHS.
5. Specialization of a condition by adding a new variable constraint.
6. Specialization of a condition by replacing a relation or template functor by a more specialized functor.
7. Decreasing a rule salience, so that a given assertion will not take place.

4.4 Execution Graphs

An *execution graph* is a graph which captures the structure of an expert system's rule base. The graph has two types of nodes: rule nodes and assertion nodes. Rule nodes stand for rule firings and assertion nodes stand for facts asserted to the working memory. Rule nodes are connected to assertion nodes that represent the assertions against which the rule's conditions are matched. The rule nodes are ordered according to the order on which they are fired. In Figure 2 there is a part of the execution graph of the auto rule bases introduced above. This part of the execution graph conforms to the case in which the user has asserted that the engine does start but does not run normally and the engine is sluggish. Ovals represent assertion nodes and rectangles represent rule nodes. We can see for instance that the rule node determine-sluggishness is connected to two assertion nodes, not(repair ?) and (working-state engine unsatisfactory) since these are the assertions which match that conditions of the determine-sluggishness rule. Each of the nodes in the execution graph is assigned a *confidence value* which represents the expert's belief that the element of that node (assertion or rule) need no be revised.

In the following section we will provide a description of the revision algorithm used in the FRST system. This algorithm is based on the PTR algorithm [Feldman 1993, Feldman et al 1994] for revision of logical domain theories.

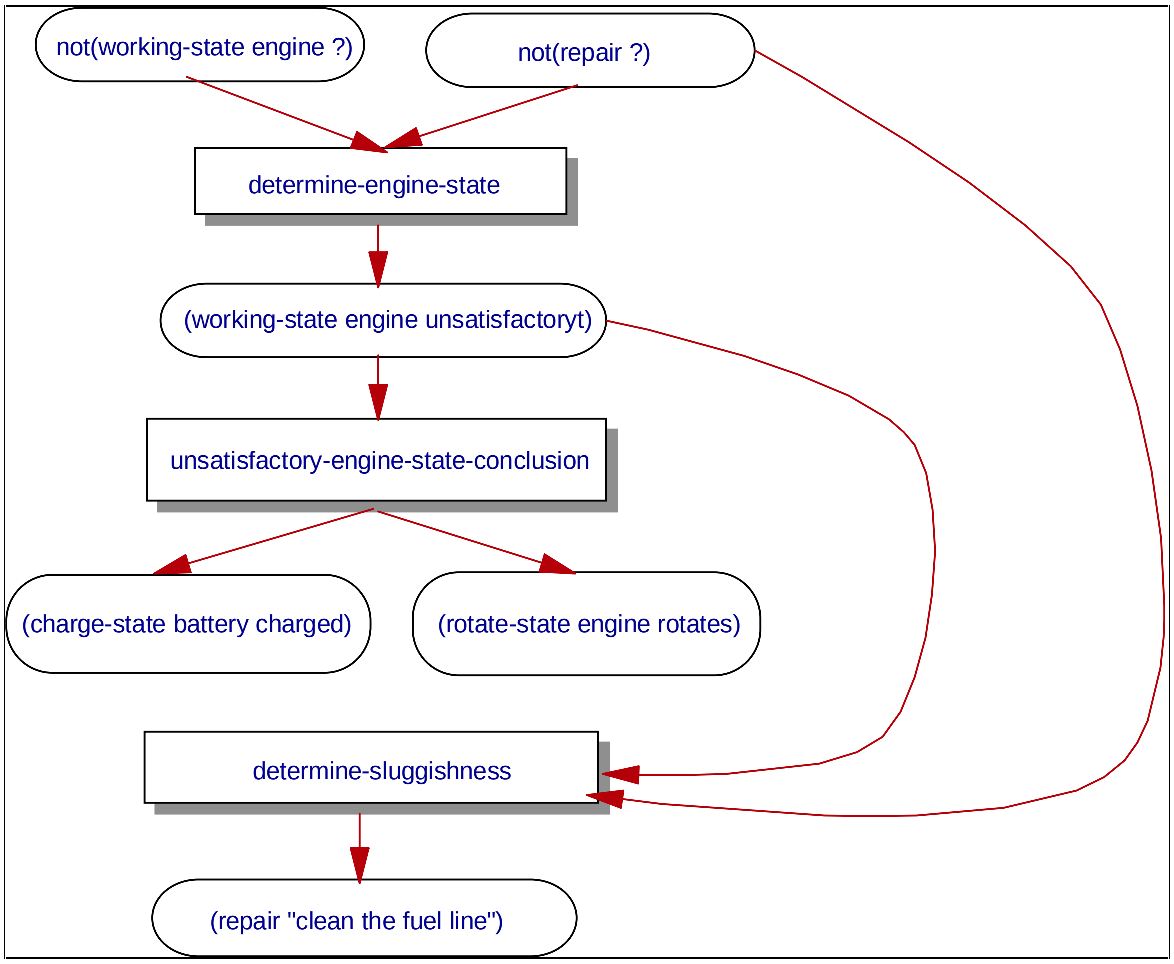

not(working-state engine ?)
not(repair ?)
determine-engine-state
(working-state engine unsatisfactoryt)
unsatisfactory-engine-state-conclusion
(charge-state battery charged)
(rotate-state engine rotates)
determine-sluggishness
(repair "clean the fuel line")

Figure 2 - An execution graph

5. The Revision Algorithm

The algorithm operates in two stages; in the first stage, it identifies flawed elements of the rule base on the basis of the cases provided, while in the second stage the element so identified are repaired. A full description of the PTR can be found in (Koppel et al, 1994). Here we provide only a short description, starting with an informal definition of an annotated rule base.

5.1 Annotated Rule bases

An annotated rule base is simply a rule base where each element *e* of the rule base (i.e., each condition, assertion or a rule) has attached to it a value P(*e*) between 0 and 1. This value, called a confidence value, is intended to reflect the user's confidence in that particular rule base element. It is a form of bias, and it is used by the revision algorithm to guide the process by which flawed elements of the rule base are identified.

The intuition is that — if the user is very confident that a particular element of the rule base does not need to be deleted or repaired — then they will assign a value close to 1. On the other hand, if the user is less certain of a given element's correctness, it will be assigned a value closer to 0. Elements assigned a value of 1 are immune to revision, while elements assigned a value 0 are removed from the rule base. In practice, of course, confidence values are generally assigned according to some default assignment scheme, with the user being expected to override the explicit confidence values as necessary.

5.2 Locating Flawed Elements

In order to locate flawed elements, the algorithm first recasts the annotated rule base as an execution graph. Each confidence value is mapped to an edge of the execution graph.

Once the execution graph is constructed, the algorithm examines each known case in some random order and uses it to update the confidence values in the execution graph. Briefly, this process consists of first computing a numeric notion of proof "flow" through the rule base, comparing the value obtained with the correct execution of the case, and then sweeping the resulting error term back through the graph to update the confidence values. Once updated confidence values are computed, they are compared against some prespecified revision threshold. Any rule base element whose confidence value falls below this threshold is selected for revision.

While this description is more intuitive than definitive (the details of the update process appear in Feldman 1993 and in Koppel et al, 1994), it is important to note that — unlike some other theory revision algorithms — this procedure does not require enumerating possible proofs of the given case; rather, it is quite efficient, and can be performed in linear time with respect to the size of the rule base.

5.3 Revising Flawed Elements

Once an element has been selected for revision, several actions that can be taken. First, the element might be simply deleted from the rule base. Second, the element might be "repaired" either by splicing in additional structure at that element or by replacing some of its components. In addition there are some other revision operators that can be used depending on the type of the element. For example, if the flawed element is a rule, additional conditions might be added to its LHS, or some actions might be added to is RHS. Deciding what conditions or actions should be spliced in is left to an inductive learning component, such as CHAM (Kijsirikul et al, 1991) which is a typed version of FOIL (Quinaln 1990).

Of course, deleting an element or adding new conditions to some rule's RHS might affect the execution of other cases. In deciding which operator to apply one must consider the possible collateral effects of the operation. In order to make the determination of how to revise an element *e*, the revision algorithm computes two subsets of the cases {C_1, C_2, ... C_n} corresponding to those cases for which *e* currently contributes to the correct execution (the needed set, denoted N), and those cases for which *e* contributes to the incorrect execution (the obstructed set, denoted O). Based on the composition of these sets, the revision algorithm decides whether to delete or repair the element in question.

6. The FRST System

In this section we will describe the FRST system architecture and will describe the operations the user can perform in a typical session with the system. The user provide to the system a list of annotated cases and an *a priori* bias that will be used in order to select specific revision operator out of the variety of the revision operators capable of fixing a given flaw. In Figure 3 we can see a block diagram of the FRST system. The system can work in two modes, one the automatic mode where the system takes into account the user's bias encoded in the rule base (described in the next subsection) and revise the rule base so that all known cases are executed correctly. The other mode is an interactive mode (similar to (Nédellec and Causse 1992)) where the user can select various revision operators and examine how they affect the accuracy of the rule base. In Figure 4 we can see the main screen of the revision module. The screen shows the execution graph of case #24 and provides the user several push button for performing actions related to the revision of the rule base.

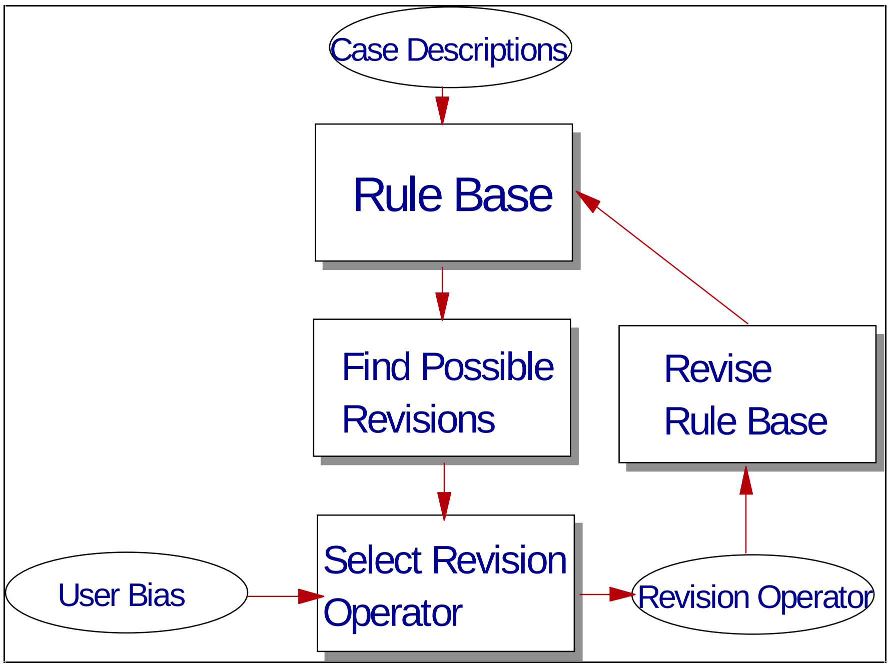
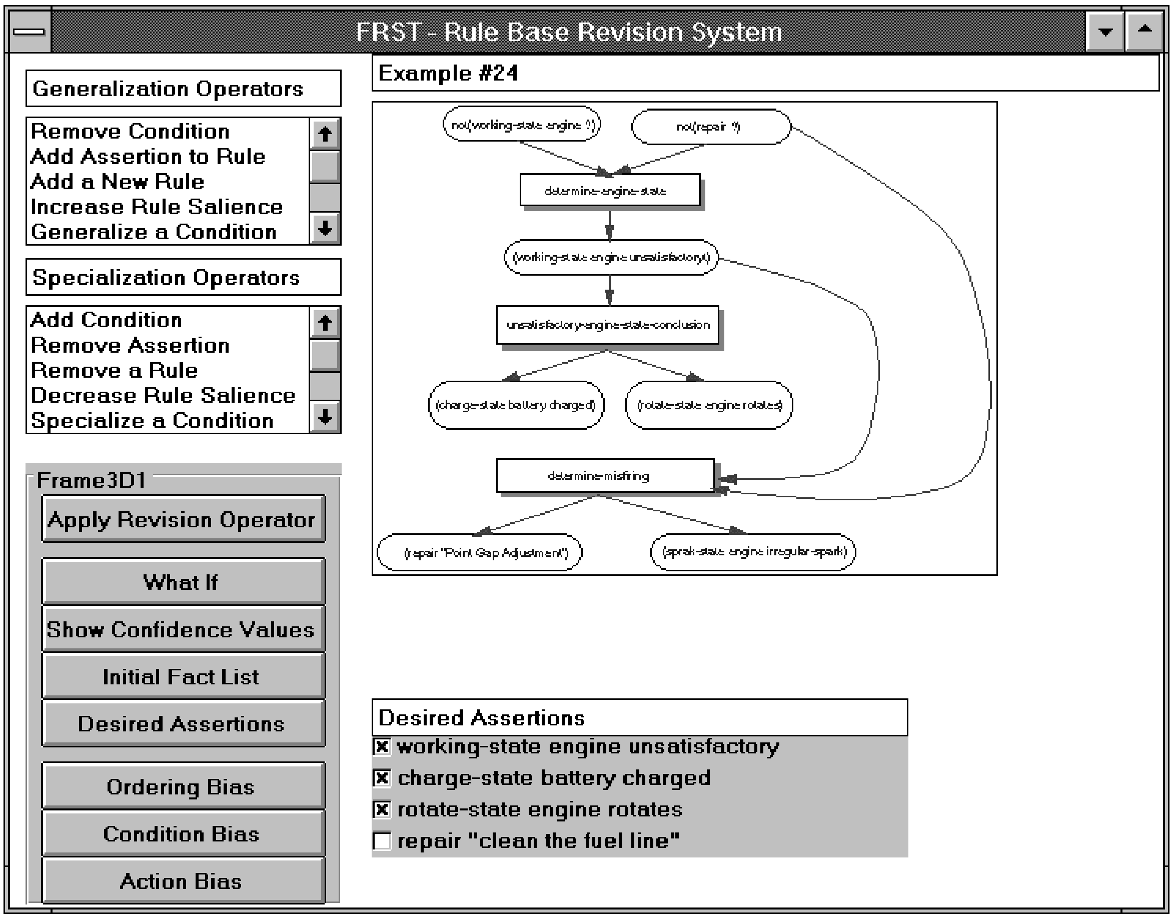


Figure 3- Block diagram of FRST

In the table 1 we list all the actions that can be performed by the user in this module.

Table 1 - Available Commands in the Revision Module of FRST

Command	Meaning
Apply Revision Operator	Apply the selected revision operator on the rule base
What if	Test the effect that a given revision operator has on the rule base (with regard to the accuracy of the rule base on the known cases)
Show Confidence Values	Annotates the execution graph with the confidence values
Initial Fact List	Show the list of facts known to be true (with the answers provided by the user) for the specific case
Desired Assertions	Show the list of assertions that should be true after the case was executed
Ordering Bias	View and edit the ordering bias
Condition Bias	View and edit condition biases (local and global)
Action Bias	View and edit action biases (local and global)



Desired Assertions

- [x] working-state engine unsatisfactory
- [x] charge-state battery charged
- [x] rotate-state engine rotates
- [ ] repair "clean the fuel line"

Figure 4 - Selection of revision operator to solve a flaw

The user can test the effect of any of the revision operators by examining the accuracy of the revised rule base on all the known cases. When the user is confident about the revision operator that should be applied he just selects it and presses the “Apply Revision Operator” button.

6.1 A Bias Language

In this subsection we will describe the bias language used for annotating rule bases. This form of bias is used by the FRST system in order to select the appropriate revision operator. The general structure of the rule base is as follows:

- Rule Section
- Global Ordering Bias
- Global Condition Bias
- Global Action Bias

The first section contains all the rules and after that we have three sections that define global biases which are related to clusters of elements and not to specific rules. We will elaborate on each bias section later and explain how the clusters are organized.

Each rule is being rewritten in the following format

(defrule rule-name
  (declare (salience S))
  ordering bias
  (condition | *condition bias*)*
  RHS bias
  =>
  (action | *action bias*)*
  LHS bias
)

All biases are optional *i.e.*, some or all the biases of a rule may be omitted. We will move now to a detailed description of each of the biases.

6.1.1 Ordering Bias

The ordering bias provides a partial order on the rules, any modification to the salience of any rule should conform to that partial order. As with all other bias types, if the user does not have any preferences on the ordering of the rules, the ordering bias can be omitted. In addition to providing ordering bias on an individual rule level, the user can define a rule hierarchy. At each node in the hierarchy there is a rule cluster. At the leaves of the hierarchy we have individual rule names. The user can provide ordering bias between individual rules or rule clusters to other individual rules or clusters. Ordering bias for individual rules is specified at the rule level and ordering bias from larger clusters (clusters that contain more than 1 rule) is specified at the global ordering bias section of the rule base. In the following table we will treat individual rules as clusters of size 1, so all ordering biases will be specified between clusters. All operators are relative operators. It does not make sense to specify absolute salience for clusters since the only effect that these salience have is the partial order they provide to rule execution. In Table 2 we can see the possible keywords used in specifying the ordering bias.

Table 2 - Ordering Bias

Keyword	**Argument**	**Meaning**
Highest		this cluster has the highest salience
Lowest		this cluster has the lowest salience
Higher than	Cluster1	the salience of rules contained this cluster is higher than rules contained in Cluster1.
Lower than	Cluster1	the salience of rules contained this cluster is higher than rules contained in Cluster1.

Example:

One of the common techniques in rule base programming is to use phases, where we move from phase to phase by using control assertions. Each phase is performed by a set of rules, where there is some partial order between the rules in the set. In a way we can view the global ordering in the rule base as ordering in two levels. The first level is between rule sets and the second level is within each rule set. This conform to a two level hierarchy of rule clusters, where in the first level we have the rule sets and a partial order on them and then in the next level we have singleton clusters which are partially ordered within the cluster.

6.1.2 Condition Bias

Providing revision bias on an element by element basis might be unreasonable when dealing with large knowledge bases. We propose using an assertion hierarchy which will permit sharing of bias across portions of the rule base. The hierarchy classifies conditions according to their meaning (e.g., colors, shapes etc.). All such elements would be assigned identical revision bias. The biases provided for the different condition clusters form an inheritance hierarchy. An element will be affected by the bias declared for the most specific clusters that contains it. If we want to give a specific condition a special bias, we will the bias to the condition within the rule in which it is declared. All other condition biases are specified in the global conditions bias in the rule base.

For each condition cluster we will specify the preferred operators under different circumstances. For each operator we will specify the preconditions under which this operator should be performed. The first operator whose all preconditions will be satisfied will be performed. The preconditions can be classified into several categories:

1. preconditions related to the cases that are affected by the specific condition. The cases affected will determine if we want to generalize the condition or to specialize it. If there is a case for which this rule is executed and shouldn’t be executed (either shouldn’t be executed at all, or shouldn’t be executed at this point) then the condition should be specialized. Cases like that are called *obstructive cases* and are denoted as *O*. On the other hand, if there is at least one case for which this rule should have been executed at this point but wasn’t executed, then the condition

should be generalized. Cases like that are called *needed cases* and are denoted as *N*.

2. preconditions related to the semantics of the conditions. For instance we can specify a uniform bias to all conditions related to geometric figures.
3. preconditions related to the topology of the rule base. These precondition are related to the location of the condition in the execution graph of the rule base. Examples of preconditions of this kind are: some comparison on depth(condition) which is the minimal length of the path from any conclusion (final assertion) to the condition, and some comparison on height(condition) which is the minimal length from any primitive assertion to the condition.

Table 3 lists some of the operators related to the revision of condition clusters with some possible preconditions.

Table 3 - Sample Condition Bias

Revision Operator	**Precondition**	**Meaning**
Delete	*O* is empty, *N* is not empty	If no case will be executed incorrectly by deleting the condition then go ahead and delete it
Drop Constraints	*O* is not empty, *N* is not empty	Generalize a condition by dropping some of the constraints of the conditions.
Generalize Condition	*O* is not empty, *N* is not empty	Generalize the condition by climbing in the assertion hierarchy.

6.1.3 LHS Bias

This bias direct the system when we need to add new conditions to the LHS of the rule. In table 4 we can see a sample of the operators with possible preconditions that are related to addition to the rule's LHS.

Table 4 - Sample Operators for additions to the LHS

Revision Operator	**Precondition**	**Meaning**
Add Condition (automatic)	*O* is not empty, *N* is not empty	Consider all affected cases and add new conditions to the rule LHS (using CHAM)
Add Condition (biased)	*O* is not empty, *N* is not empty	Consider all affected cases and add new condition from a given list of conditions. This operator gives extra guidance to CHAM as to what condition should be considered for addition.

6.1.4 Action Bias

This bias is related to the actions of the rules. Like in the condition bias we will use the assertion hierarchy in order to specify uniform bias to clusters of actions. Clearly the action bias attached to the assertion clusters (which are in this context action clusters) will be different from the condition bias attached to the same clusters. The revision operators will be different, but the same set of preconditions used in the condition bias will be used also for the action bias. We have two major kinds of actions: assert and retract. In table 5 we can see a sample of the operators and their preconditions.

Table 5 - Sample Action Bias

Revision Operator	**Precondition**	**Meaning**
Delete	*O* is not empty, N is empty	If no case will be executed incorrectly by deleting the assertion then go ahead and delete it.
Replace Action	*O* is not empty, *N* is not empty	replace an assertion by another assertion, or replace retraction by another retraction.
Specialize Action	*O* is not empty, *N* is not empty	Specialize the action by climbing in the assertion hierarchy.

6.1.5 RHS Bias

This bias directs the system when we need to add new actions to the RHS of the rule. In table 6 we can see a sample of the operators with possible preconditions that are related to addition to the rule’s RHS.

Table 6 - Sample Operators for additions to the LHS

Revision Operator	**Precondition**	**Meaning**
Add Assertion	*O* is empty, *N* is not empty	If no case will be executed incorrectly by adding the assertion then go ahead and add it

6.1.6 Global Biases

There are three types of global biases. First, we have the section that specifies partial order between clusters of rules. This bias is very similar to the ordering bias at the rule level. The second and third global bias sections are related to the addition of new rules. The second section specifies the bias for the selection of the conditions of those rules and the third bias specifies the bias for the selection of the actions of those rules. These biases change the syntactic nature of the rule induction algorithms by specifying a different order of condition and action selection.

7. Conclusions

This paper presents a system for performing incremental revision of real rule bases based on explicit bias. We have introduced a modular architecture for specifying revision bias, and have described a family of revision operators and preconditions needed for their appropriate application. Biases can be specified to clusters of elements as well as to individual elements. We have developed a special bias language tailored specifically for use in rule bases taking into account the forward chaining inference mechanism and the exact syntax of the rules. the system can take advantage of the provided bias and converge faster to the intended rule base, however even in the absence of such bias it can revise the rule base to conform with all known cases. In the absence of any bias the system will take the most conservative path by selecting operators which change the rule base as little as possible.

This paper is a further step towards building powerful bias-guided revision systems that can be used for revision of commercial rule base applications. In our case the bias comes from the operator bias attached to clusters of rule base elements. There are certainly other forms of biases which might be used to construct efficient systems that perform accurate revisions, but we believe that those can be easily integrated into the proposed system. The system uses the PTR algorithm for identifying flawed rule base elements, however, it should be noted that the scheme of the operator bias presented here can be adopted by any incremental theory revision algorithm.

Bibliography

Cain. T., "The DUCTOR: A Theory Revision System for Propositional Domains". Proceedings of the 8th International Workshop on Machine Learning, 485-489, Evanston, IL, 1991.

Feldman R., Segre A. and Koppel M., "Incremental Refinement of Approximate Domain Theories". Proceedings of the 8th International Workshop on Machine Learning, 500-504, Evanston,IL,1991.

Feldman. R., "Probabilistic Revision of Logical Domain Theories". Ph.D. Thesis, Computer Science Department, Cornell University, Ithaca NY, February 1993.

Feldman R. and Nedellec C. "A Framework for Specifying Explicit Bias for Revision of Approximate Knowledge Bases," 7th International Workshop on Knowledge Acquisition, Banff, Canada, Feb. 1994.

Ginsberg. A., "Automatic Revision of Expert System knowledge bases". Pitman, London, 1988.

Kijsirikul B., Numao M. and Shimura M., "Efficient Learning of Logic Programs with Non-Determinate, Non-Discriminating Literals". Proceedings of the 8th International Workshop on Machine Learning, 417-421, Evanston, IL, 1991.

Koppel M., Feldman R. and Segre A. "Bias-Driven Revision," Journal of Artificial Intelligence Research, Vol. 1, pp. 159-208, 1994.

Mahoney J.J. and Mooney R.J., "Combining Neural and Symbolic Learning to Revise Probabilistic Rule Bases". Advances in Neural Information Processing Systems, Vol. 5. Morgan Kaufman, San Mateo, CA, 1993.

Murphy P.M. and Pazzani M.J., “Revision of Production System rule-bases” Proceedings of the 11th International Conference on Machine Learning, , New Brunswick ,NJ, 1994.

NASA: Software Technology Branch., CLIPS Users Guide, CLIPS Version 6.0. Lyndon B. Johnson Space Center, 1993.

Nédellec C. and Causse K., "Knowledge Refinement Using Knowledge Acquisition and Machine Learning Methods", Current Developments in Knowledge Acquisition: EKAW-92, T. Wetter & al Eds., Springer-Verlag, Mai 1992.

Nédellec C., "How to Specialize by Theory Refinement", in proceedings of the 10th European Conference on Artificial Intelligence (ECAI-92), Aug. 1992.

Nédellec C., and Rouveirol C.,. "Hypothesis Selection Biases for Incremental Learning", in Proceedings of the AAAI Spring Symposium on Training Issues in Incremental Learning, Stanford University, March 1993.Ourston. D., Ph.D Thesis. University of Texas at Austin. 1991.

Ourston D. and Mooney R. J.,"Changing the rules: A comprehensive approach to theory revision". Proceedings of the eighth National Conference on Artificial Intelligence, pages 815-820, Boston, MA, 1990.

Pazzani M. and Kibler D., "The Utility of Knowledge in Inductive Learning". Technical Report, University of California,Irvine, 1990.

Quinlan J.R., "Induction on Decision trees". Machine Learning, 1, 81-106, 1986.

Quinlan J.R., Learning Logical Definitions from Relations. Machine Learning, 5, 239-266.

Richards B.L., and Mooney R.J., "First-Order Theory Revision". Proceedings of the 8th International Workshop on Machine Learning, 447-451, Evanston,IL,1991.

Towell G.G., Shavlik J. and Noordewier M.O., "Refinement of approximately correct domain theories by knowledge-based neural networks". Proceedings of the Eighth National Conference on Artificial Intelligence, 861-866, Boston, 1990.

Wogulis J., "Revising Relational Domain Theories". Proceedings of the 8th International Workshop on Machine Learning, 462-466, Evanston,IL,1991.