Towards Friendly Concept-Learners.

Luc De Raedt Maurice Bruynooghe
Department of Computer Science
Katholieke Universiteit Leuven
Celestijnenlaan 200A
B-3030 Heverlee, Belgium

Abstract

We give a general overview of CLINT, a user-friendly interactive concept-learner, which can be used as a module for Learning Apprentice Systems. CLINT combines several interesting features : it uses domain-knowledge, generates examples, copes with indirect relevance, shifts its bias, recovers from errors and identifies concepts in the limit.

1 Introduction.

One of the achievements of early work on machine learning [Michalski *et al.*, 83] was the development of concept-learners which operated in isolation, relied heavily on the user and addressed only well defined and restricted tasks. Today, a new generation of learning systems, named Learning Apprentice Systems (LAS), is being developed, e.g. DISCIPLE [Tecuci, 88] learning preconditions of actions and object models, BLIP [Morik, 89] modelling the world by observation, PRODIGY [Minton, 88] and LEAP [Mitchell *et al.*, 85] improving their problem-solving performance, ... Concept-learning is often a subtask in this type of systems; it has to use the available knowledge and keep the user's involvement to a minimum.

This paper gives a general overview of CLINT (Concept-Learning in an INTeractive way), a user-friendly concept-learner. It uses knowledge, generates examples, copes with indirect relevance, shifts its bias, recovers from errors and identifies concepts in the limit. All these properties make it well suited as a module in a LAS. A more thorough discussion of CLINT w.r.t. example generation and assimilation can be found in [De Raedt and Bruynooghe, 88a] and w.r.t. bias, language and explanation in [De Raedt and Bruynooghe, 88b]. The contribution of this paper is that it shows how the techniques of [De Raedt and Bruynooghe, 88ab] can be integrated in one system.

The paper is organized as follows : in section 2, the learning task is formulated in a logical framework; the following section contains a full description of the learning algorithms in CLINT; an example illustrating the technique is given in section 4 and finally in section 5, we briefly touch upon related work. The appendix contains a summary of CLINT's algorithms.

2 Problem-Specification.

Concept-learning refers to the process of generating a concept-description from examples. We use a logical framework [Genesereth and Nilsson, 87]. A *concept* is a *predicate*. A *concept-description* is a set of *Horn-Clauses* defining the predicate and *examples* are ground instances of a predicate. We restrict ourselves to function free predicates. Moreover, literals are of the form $p(X_1, \ldots, X_n)$ with the X_i distinct variables. This is not an additional restriction because constants are allowed in the knowledge-base and an equality predicate is also available e.g. q(a,X,X) rewrites to q(X,Y,Z) :- A(X), Y=Z where A(a) is in the knowledge-base.

In practice the concept-learning process is not only driven by training-examples but also by bias [Utgoff and Mitchell, 82], which is -following Utgoff- anything which influences how the concept-learner draws inductive inferences based on the training-examples. The significance of bias for concept-learning is widely recognized [Utgoff and Mitchell, 82], [Utgoff, 86]. Although bias may be incorporated in a concept-learning system in a variety of ways, we will only consider bias in the form of restrictions on the description language.

In our framework, a bias is *correct* for learning a predicate p if there exists a set of Horn-clauses S for the predicate p in the description language associated with the bias, such that all positive and no negative examples are covered by S. A concept is *conjunctive* if it can be defined with one Horn-clause. Otherwise it is *disjunctive*.

The knowledge-base, employed by the learner, contains two different kinds of predicates. *Basic* predicates are defined by the user and assumed to be correct. They are specified by arbitrary Horn-clauses. *Learned* predicates are defined by the learner and derived from opportunities. An *opportunity* occurs when the user signals that there is an error in the learner's knowledge-base. In our framework, an opportunity takes the form of an uncovered positive or a covered negative example. Horn-clauses for learned predicates satisfy one of the language-schema's in the learner's knowledge-base. A *language-schema* describes a set of syntactic restrictions, i.e. a

bias, which must be satisfied by a Horn-clause in order to belong to the language associated with the schema. Our concept-learner has an ordered series of language-schema's $L_1, L_2, \ldots, L_n, \ldots$

For each opportunity, the learner analyzes and augments its knowledge by asking membership questions to the user. Each question asks for the classification of an example for a learned predicate. By processing the opportunities, the learner's knowledge base gradually converges to the desired one.

By now we can define our notion of concept-learning more precisely :

Given :

- a knowledge-base of basic and learned predicates
- for learned predicates, a set of positive and negative examples
- a series of language-schema's, $L_1, L_2, \ldots, L_n, \ldots$
- an opportunity to learn, indicated by the oracle
- an oracle, willing to answer membership questions

Find :

- an adapted knowledge-base, such that all positive examples and no negative examples are covered

3 Processing an opportunity.

The two classes of opportunities, uncovered positive examples and covered negative examples, are processed in a different way.

3.1 Processing a positive example.

When the learner receives an uncovered positive example, it constructs a new clause covering the example and adds it to the knowledge-base. To compute the clause, the learner constructs an explanation (cf. later) in order to obtain a starting clause, and generalizes it by asking questions. The algorithm to process an uncovered positive example is shown in `handle_uncovered` of the appendix.

As examples are ground facts, they only specify the objects which are directly involved and not the relevant relations between these objects with respect to the concept. All relations in the knowledge-base are *potentially* relevant. However, considering all these relations simultaneously is clearly not feasible since this would very rapidly lead to combinatorial problems. Therefore, our learner divides its knowledge-base into different regions and at any time only relations belonging to one region are considered. The regions are computed from the example, the knowledge-base and the language-schema's. Given an example, a knowledge-base and a language-schema, a maximal[1,2] set of relations holding for the example and satisfying the schema, is computed. This set constitutes the *explanation*. From the explanation and the example, a starting clause is derived by turning all constants into distinct variables in the clause with as head the example and as body the explanation. Note that the starting clause for an example is true, provided that the concept can be described in terms of the predicates in the knowledge-base and the language-schema. The reason is that a starting clause is the most specific one covering the example with respect to the knowledge-base and the language-schema. Since the language-schema's are ordered according to growing expressiveness of the languages they describe, we can order the starting clauses according to the language they belong to. Within this order, the first starting clause, which does not cover any negative example, is passed to the generalization procedure.

In the next step, the starting clause is carefully generalized using a specific to general search which drops multiple conditons (this part constitutes the `while` loop). The generalization procedure computes a new clause obtained from the old clause (the starting clause), by deleting a subset S of the body of the old clause, such that the new clause covers an example e which was not covered by the old clause, the new clause does not cover negative examples and S is minimal. The example e is then presented to the user for classification. If it is positive the old clause is replaced by the new one, and the process is repeated. Otherwise, the learner backtracks in order to construct another subset. If there are no more subsets fulfilling the conditions, the generalization process terminates and the clause is added to the knowledge-base. During generalization, the system may have collected new negative examples. The algorithm tests whether these are covered by previously learned rules, and if necessary, it takes appropriate action in order to recover from possible errors.

3.2 Processing negative examples.

When the learner receives a covered negative example, there must be an incorrect clause in its knowledge-base. The algorithm to handle covered negative examples is shown in `handle_covered` in the appendix. A clause c is *incorrect* if there exists a substitution θ such that head(c)θ=false and body(c)θ is true. To locate incorrect clauses, the learner constructs the proof tree explaining why the negative example is covered and analyses it. If necessary it asks intelligent questions to the user. These questions are also membership-questions. They ask for the classification of examples which correspond to nodes in the proof tree. The method to locate incorrect clauses is very similar to Shapiro's debugging method used in the Model Inference System [Shapiro 82]. When an incorrect clause is located, it is retracted from the knowledge-base. Of course, in order to maintain consistency the learner must verify whether the positive examples which were covered by the incorrect clause, are still covered. If not, it will generate and process an opportunity for each uncovered positive example. Also, if a predicate was learned after the predicate for the incorrect clause and if the latter one was involved in learning the former, then

[1] As such a set may indicate why an example belongs to a concept, we call it an explanation However, in contrast to EBG [Mitchell et al., 86], our notion of explanation does not mean that the conditions are sufficient.

[2] For simplicity, we will assume that the explanation is unique. In [De Raedt and Bruynooghe 88b], we show that the explanation is unique for some interesting languages and show how to extend the method if explanations are not unique.

we must also verify its positive and/or negative examples. In order to avoid these interactions between different predicates as much as possible, it is advised that predicates are thoroughly tested before other ones are learned. The complete algorithm is shown in the appendix.

3.3 About the algorithm.

At this point we want to stress that the subsets S in `handle_uncovered` can efficiently be computed using a breadth-first search, pruned by knowledge and guided by heuristics. For more details see [De Raedt and Bruynooghe, 88a].

The efficiency of the algorithm depends on the redundancy in the starting clause(s) and the size of the knowledge-base. For a lot of interesting problems, the needed resources are acceptable (cf. [De Raedt and Bruynooghe, 88a]). The number of questions generated by the algorithm, depends on the number of conditions and the redundancy in the starting clause(s). In general, the number of questions needed is small (cf. [De Raedt and-Bruynooghe, 88a]).

Because of space-limitations we do not discuss the language-schema's and the way they are implemented here. For more details see [De Raedt and Bruynooghe, 88b]. Instead, we give a detailed example.

CLINT's algorithm has several interesting features :

- It copes with *indirect relevance* [Michalski, 83] because the user only has to supply the directly involved objects in an example and not the important relations holding between them. This is a more natural and less demanding way to describe examples.
- Disjunctive as well as conjunctive predicates can be learned.
- The system can shift its bias when it discovers that its language is not sufficient to describe the predicate. This situation occurs whenever the starting clause covering a positive example in a language covers negative examples (see also the example). In that case the language of the clause will be shifted. So, CLINT shifts the bias at the clause-level.
- The system generates most of the examples it needs. The user has to specify at most one positive example to learn a clause.
- The system is a closed-loop one. This means that newly learned predicates are integrated (assimilated) in the knowledge-base and that once learned, they can be used like any other predicate.
- The system provides a kind of error-recovery. If an incorrect clause is assimilated, the system will retract it when there is an opportunity to do so.
- CLINT identifies concepts in the limit, provided that the predicate can be described in one of the languages and the current knowledge. This limiting behaviour is proved in [De Raedt and Bruynooghe, 88ab]. For conjunctive predicates and simple languages there is even finite identification.

There are still some remaining problems with the algorithm :

- Recovering from an error in an assimilated predicate may also affect other predicates. Therefore, recovering from such errors may involve a lot of work.
- The order in which predicates are learned is important. Ideally, the easier predicates should be learned first, and the more difficult ones, which use the easier ones in their definition should only be learned afterwards. However, if the user presents the predicates in a different order, this does not necessarily lead to problems, because of CLINT's error-recovery ability. In fact, problems only arise when there is a positive and a negative example such that given the knowledge and the languages, there is no clause which covers the positive example and which does not cover the negative one. In that case, CLINT will keep on shifting its bias.
- The learned clauses depend very much on the knowledge CLINT possesses. If CLINT knows all relevant predicates w.r.t. the concept to be learned, then CLINT will learn a small number of clauses in easy languages. On the other hand, if CLINT knows only predicates that are not so relevant, CLINT will learn a larger number of clauses or the bias of the clauses will be more complex. This problem seems to be inherent to concept-learners.
- CLINT does not take into account the relationship between different clauses for the same concept. As a consequence, CLINT does not always learn the concept-description whith the minimal number of clauses. This problem also arises in MARVIN [Sammut and Banerji, 86] and MIS [Shapiro, 82].
- For some language-schema's, knowledge-bases and examples there may be more than one explanation. If the number of possible explanations is large, the algorithm may require a lot of computation. For a discussion of these issues and some possible solutions we refer to [De Raedt and Bruynooghe, 88b].

4 An example session.

The knowledge-base consists of the following definitions in PROLOG :

```
place(p1).   composition(p1,concrete).   color(p1,white).
place(p2).   composition(p2,sand).       color(p2,blue).

table(t1).   composition(t1,metal).      color(t1,red).
table(t2).   composition(t2,wood).       color(t2,grey).

block(b1).   composition(b1,wood).       color(b1,red).
block(b2).   composition(b2,plastic).    color(b2,blue).
block(b3).   composition(b3,metal).      color(b3,red).
block(b4).   composition(b4,stone).      color(b4,grey).
block(b5).   composition(b5,glass).      color(b5,white).

weight(t1,50).     weight(t2,50).
weight(b1,20).     weight(b2,40).   weight(b3,30).
weight(b4,120).    weight(b5,40).

object(X) :- place(X).
object(X) :- table(X).
```

object(X) :- block(X).

less(X,Y) :- X < Y.

Comments are in italic, user input in bold and CLINT's output in roman.

?-learn.
Positive Example of New Predicate ?
> same_color(t2,b4).
An explanation for the example is computed using the first language schema : table(t2), block(b4), object(t2), object(b4). The starting clause is derived from the explanation. Notice that the starting-clauses contain only relations which hold for the example. The first language-schema requires that all variables in the body of the starting-clause also occur in the head of the starting-clause. Although the system knows that object is more general than block and table, it keeps these predicates in the body in order to facilitate the generalization procedure. This knowledge is however used by the system during the generalization process and in order to simplify final clauses.
Starting clause :

```
same_color(X,Y)  :-
  table(X), block(Y), object(X),
  object(Y) .
```

The system tries to generalize and after considering one candidate generalization, it constructs an example and a subset fulfilling the necessary conditions, so it asks the user to classify the example. The example is choosen randomly from the set of examples which can be constructed using the subset.
Is same_color(t1,t1) positive ? [y/n]
> y.
After searching 1 clauses, next clause :

```
same_color(X,Y)  :-
  table(X) , object(X) , object(Y).
```

As the user classifies it as positive, the system generalizes.
Is same_color(b1,t1) positive ? [y/n]
>y.
After searching 2 clauses, next clause :

```
same_color(X,Y)  :-
  object(X) , object(Y) .
```

The system is unable to find further generalizations, so the clause is asserted in the knowledge-base.
No more valid generalizations after searching 2 clauses
Generalized clause :

```
same_color(X,Y)  :-
  object(X) ,  object(Y) .
```

?-error.
Example which is incorrectly handled ?
> same_color(t2,b5).
The user signals that an error has occurred. The system locates and retracts the incorrect clause by analyzing the proof for the example. If object was a basic predicate, then the two verification questions would not be asked since it is assumed that basic predicates are correct.
Verify whether object(b5) is true ? [y/n]
> y.
Verify whether object(t2) is true ? [y/n]
> y.
Retracting wrong clause :

```
same_color(X,Y)  :-
  object(X) , object(Y) .
```

same_color(t2,b4) is positive and uncovered
Starting clause for same_color(t2,b4) covers negative example same_color(t2,b5)
As the system discovers that the starting clause covers a negative example, it decides to shift the bias of this clause. The new language-schema includes clauses where each condition can have one variable not occurring in the head. No relations between these existentially quantified variables are allowed.
Shifting bias, new starting clause :

```
same_color(X,Y)  :-
  table(X), block(Y), weight(X,XW),
  weight(Y,YW), composition(X,XC),
  composition(Y,YC), color(X,CX),
  color(Y,CY), object(X), object(Y).
```

Starting clause for same_color(t2,b4) covers negative example same_color(t2,b5)
The new language-schema is the same as the old one except that it allows for relations between the existentially quantified variables.
Shifting bias, new starting clause :

```
same_color(X,Y)  :-
  table(X), block(Y), weight(X,XW),
  weight(Y,YW), composition(X,XC),
  composition(Y,YC), color(X,CX),
  color(Y,CY), object(X), object(Y),
  CX = CY, less(XW,YW).
```

Is same_color(b1,b3) positive ? [y/n]
> y.
After searching 2 clauses, next clause :

```
same_color(X,Y)  :-
  block(Y), weight(X,XW),
  weight(Y,YW), composition(X,XC),
  composition(Y,YC), color(X,CX),
  color(Y,CY), object(X), object(Y),
  CX = CY, less(XW,YW).
```

I know same_color(b1,t1) is positive, so I generalize !
After searching 1 clauses, next clause :

```
same_color(X,Y)  :-
  weight(X,XW),
  weight(Y,YW), composition(X,XC),
  composition(Y,YC), color(X,CX),
  color(Y,CY), object(X), object(Y),
  CX = CY, less(XW,YW).
```

Is same_color(b1,b2) positive ? [y/n]
> n.
Is same_color(b1,b1) positive ? [y/n]
> y.
After searching 8 clauses, next clause :

```
same_color(X,Y)  :-
  weight(X,XW), weight(Y,YW),
  composition(X,XC),composition(Y,YC),
  color(X,CX),color(Y,CY),object(X),
  object(Y),CX = CY.
```

Is same_color(b2,p2) positive ? [y/n]

```
> y.
After searching 6 clauses, next clause :
   same_color(X,Y)   :-
     weight(X,XW),
     composition(X,XC),composition(Y,YC),
     color(X,CX),color(Y,CY),object(X),
     object(Y),CX = CY.
Is same_color(p1,p1) positive ? [y/n]
> y.
After searching 6 clauses, next clause :
   same_color(X,Y)   :-
     composition(X,XC),composition(Y,YC),
     color(X,CX),color(Y,CY),object(X),
     object(Y),CX = CY.
No more valid generalizations, after searching 31 clauses
Generalized and simplified clause :
   same_color(X,Y)   :-
     color(X,CX),color(Y,CY),CX = CY.
```

The final clause can be simplified using information gathered during the while-loop in the appendix. This session with CLINT, implemented in BIMprolog took about 50 seconds of CPU-time on a SUN 3/50.

5 Related work.

Our approach is related to the work on version-spaces [Mitchell, 82], learning apprentices [Tecuci, 88], EBG [Mitchell *et al.*, 86], experimentation [Subramanian and Feigenbaum, 86],[Sammut and Banerji, 86], [Krawchuk and Witten, 88], indirect relevance [Buntine, 87], identification in the limit and debugging [Shapiro, 82] and shift of bias [Utgoff and Mitchell, 82],[Utgoff, 86].

CLINT's basic algorithm is very similar to version-spaces, except that CLINT stores the negative examples instead of the G-set. For rich description languages this is often more efficient [De Raedt and Bruynooghe, 88a].

As CLINT builds plausible explanations for examples in an inductive way it attempts to overcome the strong theory requirement imposed by EBG. Therefore it also needs more than one example to find a new definition.

CLINT asks membership questions to the user, just like Factoring [Subramanian and Feigenbaum, 86], MARVIN [Sammut and Banerji, 86], ALVIN [Krawchuk and-Witten, 88]. However, none of these techniques copes with indirect relevance, because they all require that the relevant relations in an example are specified. Subramanian uses some kind of propositional logic, while we use subsets of first order logic. Factoring is more powerful but less general than CLINT because it requires that the space is factorable. For MARVIN and ALVIN it is less clear which concepts can be learned. ALVIN is less efficient and more powerful because it always tries to construct crucial objects, while CLINT constructs just significant objects. Also, MARVIN and ALVIN cannot shift their bias and only cope with a more limited form of indirect relevance (from some given relations it is possible to induce other ones).

Coping with indirect relevance in CLINT is very similar to the technique in PGA [Buntine, 87]. Only, PGA is unable to shift its bias or to generate examples.

CLINT is related to the learning component of DISCIPLE [Tecuci, 88]. However, it does not suffer from some of the problems with DISCIPLE [Tecuci, 88] like the use of a restricted versionspace approach, limited explanations and no means for error-recovery.

The debugging method used in CLINT is adapted from [Shapiro, 82], and optimized not to ask questions about basic predicates.

Previous work on shifting the bias concentrated on propositional logic [Utgoff, 86]. One of the main contributions of CLINT is the introduction of a series of languages, which are subsets of first order logic and which can be used to shift the bias. These languages are computed dynamically from one example and the knowledge-base.

6 Conclusions.

We presented an original approach to concept-learning combining several interesting features (example-generation, shift of bias, use of knowledge, indirect relevance) into one system. We believe that the resulting system CLINT, is more friendly than other concept-learners. Hence, it seems very promising for use in learning apprentice systems. CLINT is currently being integrated in an expert system shell in order to obtain a learning apprentice.

Acknowledgements.

We would like to thank Y. Kodratoff, G. Tecuci, G. Sablon and J.F. Puget for suggestions on CLINT and G. Sablon, D. De Schreye, M. Vermaut and the referees for their comments on earlier versions of this paper. Part of the work was done while Luc was a visitor in the Inference and Learning Group of Y.Kodratoff at the Universite de Paris-Sud. Maurice Bruynooghe is supported by the Belgian National Fund for Scientific Research and Luc De Raedt's visit was partially supported by the Belgian National Fund for Scientific Research by means of an IBM-travel grant.

References.

[Buntine, 87] Buntine, W., Induction of Horn-Clauses : methods and the plausible generalization algorithm, *International Journal of Man-Machine Studies*, 26(4): 499-520, 1987.

[De Raedt and Bruynooghe, 88a] De Raedt, L., Bruynooghe, M., On interactive concept-learning and assimilation, In *Proceedings of the 3rd European Working Session On Learning*, pages 167-176, Glasgow, October 1988.

[De Raedt and Bruynooghe, 88b] De Raedt, L., Bruynooghe, M., On explanation and bias in inductive concept-learning and assimilation, Report , Department of Computer Science, Katholieke Universiteit Leuven, Belgium, 1988.

[Genesereth and Nilsson, 87] Genesereth, M.R., Nilsson, N.J., *Logical foundations of Artificial Intelligence*, Morgan Kaufmann, Los Altos, 1987.

[Krawchuk and Witten, 88] Krawchuk, B., Witten, I., On asking the right questions, In *Proceedings of the 5th Machine Learning Conference,* pages 15-22, Ann Arbor, June 1988.

[Michalski *et al.,* 1983] Michalski, R.S., Mitchell, T.M., Carbonell, J.G., (editors) *Machine Learning : an artificial intelligence approach,* Tioga publishing company, Palo Alto, 1983.

[Michalski, 83] Michalski, R.S., A theory and methodology of inductive learning, In Michalski, R.S., Mitchell, T.M., Carbonell, J.G., (editors), *Machine Learning : an artificial intelligence approach,* Tioga publishing company, Palo Alto, 1983, pp. 83-134.

[Mitchell, 82] Mitchell, T.M, Generalization as search, *Artificial Intelligence,* 18(2):203-226, 1982.

[Mitchell *et al.,* 85] Mitchell, T.M., Mahadevan, S., Steinberg, L.I., LEAP : a learning apprentice for VLSI design, In *Proceedings of the ninth International Joint Conference on Artificial Intelligence,* pages 573-580, Los Angeles, California, August 1985.

[Mitchell *et al.,* 86] Mitchell, T.M., Keller, R.M., Kedar-Cabelli, S.T., Explanation based generalization : a unifying view, *Machine Learning,* 1(1): 47-80, 1986.

[Minton, 88] Minton, S., Quantitative results concerning the utility of Explanation Based Learning, In *Proceedings of the seventh National Conference on Artificial Intelligence,* pages 564-569, St.Paul, Minsota, August 1988.

[Morik, 89] Morik, K., Sloppy Modeling, In Morik, K., (editor) *Knowledge Representation and Organization in Machine Learning,* Lecture Notes in Artificial Intelligence, Vol. 347, Springer Verlag, 1989.

[Sammut and Banerji, 86] Sammut, C., Banerji, R., Learning concepts by asking questions, In Michalski, R.S., Mitchell, T.M., Carbonell, J.G., (editors), *Machine Learning : an artificial intelligence approach,* 2, Morgan Kaufmann, Los Altos, 1986.

[Shapiro, 82] Shapiro, E.Y., *Algorithmic program debugging,* The MIT Press, Cambridge, 1982.

[Subramanian and Feigenbaum, 86] Subramanian, D., Feigenbaum, J., Factorization in experiment generation, In *Proceedings of the fifth National Conference On Artificial Intelligence,* pages 518-522, Philadelphia, August 1986.

[Tecuci, 88] Tecuci, G., DISCIPLE : A theory, methodology and system of expert knowledge acquisition, Doctoral Dissertation, Universite Paris-Sud, Orsay, 1988.

[Utgoff and Mitchell, 82] Utgoff, P.E., Mitchell, T.M., Acquisition of appropriate bias for concept learning, In *Proceedings of the second National Conference On Artificial Intelligence,* pages 414-417, Philadelphia, August 1982.

[Utgoff, 86] Utgoff, P.E., Shift of bias for inductive concept-learning, In Michalski, R.S., Mitchell, T.M., Carbonell, J.G., (editors), *Machine Learning : an artificial intelligence approach,* 2, Morgan Kaufmann, Los Altos, 1986.

Appendix : a summary of CLINT's algorithm.

```
CLINT =
  As long as there are still opportunities
    Receive an example e for a predicate p  from the
    user
    Add e to the set of examples
    Handle_example(p,e)

Handle_example(p,e)=
  If e is positive and uncovered
  then Handle_uncovered(p,e)
  If e is negative and covered
  then Handle_covered(p,e)

Handle_uncovered(p,e) =
  bias_index = 0
  Repeat
    bias_index = bias_index + 1
    Find the explanation x for e in the language
    L_bias_index :
      Construct the maximal set of relations from e,
      L_bias_index and the knowledge-base
    Construct c : head(c) = e and body(c) = x
    Derive c' from c by turning all constants into dis-
    tinct variables
  Until c' does not cover negative examples
  New_negatives = ∅
  possible=true
  While possible do
    Delete a subset S from body(c') to obtain a new
    clause c'' such that :
      (1) c'' covers an example e'
      (2) c' does not cover e'
      (3) c'' does not cover any negative example
      (4) There is no subset of S for which (1,2,3) holds
    Ask the user to classify e'
    If e' is positive then c'=c''
    If e' is negative
    then backtrack on the subset S
          add e' to New_negatives
    If there are no more plausible subsets
    then possible=false
  Add c' to the knowledge-base
  If there are clauses for p different from c'  then
     For all e'' in New_negatives do
       Handle_example(p,e'')

Handle_covered(p,e) =
  Analyze the proof tree for e in order to obtain an
  example e' and an incorrect clause c such that :
    (1) e' is negative
    (2) there is a substitution θ such that head(c)θ=e'
    (3) body(c)θ is true
  Retract c
  For all positive examples e'' of  p covered by c do
    Handle_example(p,e'')
  For all predicates q learned after p and using p do
    For all examples e'' of q do
      Handle_example(q,e'')
```