Rendell, L.A. 1988. Learning Hard Concepts, European Working Session on Learning, Glasgow, Scotland, October 1988, pp. 177–200.

Rendell, L.A., Seshu, R.M., and Tcheng, D.K. 1987. Layered concept learning and dynamically variable bias management, *Proc. Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987.

Samuel, A.L. 1963. Some studies in machine learning using the game of checkers, *IBM Journal of Research and Development*, 1959, reprinted in in Feigenbaum, E.A. and Feldman, J. (Ed.), *Computers and Thought*, McGraw-Hill, pp. 71–105.

Smith, E.E. and Medin, D.L. 1981. *Categories and Concepts*, Harvard University Press.

Watanabe, S. 1969. *Knowing and Guessing: A Formal and Quantitative Study*, Wiley.

90-14 90-14

16

LEARNING EXPERT KNOWLEDGE BY IMPROVING THE EXPLANATIONS PROVIDED BY THE SYSTEM

Yves Kodratoff
(CNRS , Université de Paris-Sud and George Mason University)

Abstract

This chapter presents a framework for integrating all possible sources of explications: those stemming from explanation-based learning (EBL) and those describing the relationships of a recognition function with the positive or negative examples of the concept it is supposed to cover. For those directly obtained from a human expert, refer to Chapter 19. The deep analogy between these different explanatory sources is explained and carefully exemplified. The chapter starts with a description of what expert knowledge, a knowledge fit to symbolic approaches, is, as opposed to casual knowledge, fit to numeric approaches. Our main point is that expert knowledge is almost as much devoted to efficiency as to explication. We then explain how much the improvement and progressive refinement of a generalization depends on the quality of the explanations one can get—"why the generalization is good or bad." Finally, we propose some general methodology to improve the explanations themselves.

16.1 INTRODUCTION

The word "learning" is currently used in such a wide variety of meanings that the research area attracts people that have different or even opposite interests. Most people claiming to belong to artificial intelligence (AI) tend to be oriented towards a symbolic approach, which can be characterized by its care for providing explanations in the user's language. This demands intensive and explicit use of background knowledge and symbolic reasoning. It can possibly lead to some lack of efficiency and accuracy.

On the other hand, most other techniques, like data analysis, connectionism, operations research, and all statistical methods emphasize the needs for efficiency and accuracy. They fulfill these needs by using a knowledge representation, the semantics of which is the closest possible to the semantics of numbers. Therefore, these approaches make implicit use of background knowledge, and they usually are unable to provide explanations in the user's language.

Actually, there seems to be not only a double, but a triple misunderstanding of the expression "machine learning" (ML) that arises on the one hand, between specialists in artificial intelligence (called *AI-learnists* in the rest of this chapter) and, on the other hand, between more psychology-oriented people (called *psy-learnists* in the rest of this chapter), and non-AI-oriented learnists (called *NOT-AI-learnists* in this chapter). We shall attempt to qualify AI-learnists by two features. The first feature concerns the topic they are working on.

First Criterion **There exists a body of explicit knowledge in the field in which learning takes place, expressed in the expert's language.** Having expertise in this field means mastering the field's language (i.e., the symbolic representation of the domain knowledge) and being able to explain to another expert the reasons for one's choices. There may be some disagreement among experts, but the disagreement is over the reasons rather than the choices themselves. In everyday life, this is enough to define expert knowledge, as opposed to casual knowledge. Typically, mathematics is expert knowledge, and bicycle riding is casual knowledge.

We need to add here a second feature concerning the way these approaches are working on their topic. Within AI-learning itself many discussions take place, like the recent one between explanation-based learning (EBL) and empirical learning. These discussions tend to hide that both approaches are issued from the AI approach, and both are characterized by the following.

Second Criterion **Their results are expressed in the language of the expert him/herself.** The other approaches to ML, illustrated by statistics, and also, more recently, by the connectionist approach [Hinton—Chapter 20, this volume], whose aim is rather *efficiency and accuracy*, and where explanations hardly can be expressed in the vocabulary of the expert. More generally, all sciences aim to develop their own language with which they can describe the behavior of their observables. AI indeed develops its own vocabulary as well, but it must include the expert's language. Statistics do provide explanations of their results, but typically these reasons are expressed in terms of "quadratic squares," or other statistical concepts. They are expressed in the vocabulary of the expert in statistics, not in the vocabulary of the

expert of the field on which statistics are done. In opposition, all the AI-inspired, EMYCIN-like expert systems provide explanations in the vocabulary of the expert in the field under study even though they are well known for providing bad explanations of their behavior because of the way they combine positive and negative beliefs.

Definitions We shall say that one is doing *expert knowledge acquisition* when the two following requirements are fulfilled.

- First, the acquisition concerns a field that possesses a body of theory such that the expert in this theory can provide explanations.
- Second, the acquisition provides explanations of its behavior and uses the vocabulary of the human experts.

When one of these conditions is not fulfilled, then we define the acquisition of knowledge as *casual knowledge acquisition* (or *non-conceptual* knowledge acquisition).

Let us give three examples of NOT-AI-learning, ordered by their distance from AI-learning.

- Riding a bicycle is an example of everyday life casual knowledge, its learning therefore belongs to casual knowledge acquisition.
- A diagnosis system that would rely on thousands of clinical cases, store all of them, and provide a diagnosis by a template-matching mechanism does not provide explanations, therefore it belongs to NOT-AI-learning. It is similar to rote-learning, which clearly is of little concern to AI specialists.
- A diagnosis system that uses pure numerical techniques in order to perform its clusterings, and generate its recognition functions, cannot give explanations in the vocabulary of the expert.

There are here some (fortunate!) shadows on the limits between AI- and NOT-AI-learning, since the clustering algorithm may or may not include, as parameters, some semantics of the field. Michalski's *conceptual clustering* [Michalski and Stepp, 1983] is very typical of a numerical technique that falls into AI-learning because it can provide some explanations of its clusters in the expert's vocabulary. NOT-AI-learnists are interested in casual knowledge acquisition, because they do not mind explanations but efficiency.

Psy-learnists mind explanations but they are interested in the way humans actually store their knowledge, which seems to be very far from the way existing knowledge-based systems store it.

One necessary condition to the generation of explanations is that the system is able to *prove* that its actions obey some constraints; the quality of this proof is another matter that will be discussed later. Section 16.3 will illustrate how proving things may be a first step towards explanations. These explanations look different when coming from EBL or from empirical learning. Actually, there has been recently a considerable emphasis on the difference between the two. EBL works by using classical theorem-proving and goal-regression techniques, while empirical learning usually looks for recognition functions obtained by a generalization from a set of positive and negative examples. This chapter will show that their essential difference does not lie in using one (as generally EBL does) or several (as generally empirical learning does) examples, but in the way each uses background knowledge. It just happens that empirical learning may hide, and often has hidden, its background knowledge into the representation of the examples. When empirical learning does not use explicit background knowledge at all, our definitions place it in NOT-AI-learning. Conversely, as soon as one is willing to pay the price for an explicit representation of the background knowledge, then empirical learning is also able to provide explanations to its user. In order to avoid confusions, we shall call *learning from examples by empirical induction* the empirical learning that does not use explicitly background knowledge, and *learning from examples by constructive induction* the one with explicit background knowledge (see the first chapter of this book for precise definitions). We therefore also want to illustrate that, contrary to the "EBL versus empirical learning" way of looking at machine learning, constructive learning from examples must also provide explanations and needs some theorem proving.

EBL is born from techniques that are efficient on a very well-defined domain. For instance, the recently defined EBG (G for *generalization*), [Mitchell, *et al.*, 1986] requires both

1. a complete theory (including a definition of the concept under learning), and
2. an instance of the concept. It would be useful to be able to define generalization in a domain where the concept to be reached is still unknown, or the theory still to be completed. This has been done by Michalski [Michalski, 1983, 1984], in his formalization of generalization. Our aim is to push forward this theory, and attempt to encompass both EBL and constructive learning from examples into it, by showing that the last also should provide explanations of its generalizations. In less formalized domains, one should at least be able to take into account human experts' explanations. This last topic is discussed in Chapter 19 of this book.

In this chapter, we hope to reach two different goals. The first is to show how a formal theorem prover can help in providing explanations that improve the concept under learning, the learning being deductive (as it should be) or, a bit more unexpectedly, inductive as well. We shall differentiate the use of theorem proving done during inductive learning from the one done during deductive learning. The second goal is to show how these formalities are actually simple, and in many cases, easy to use,

once one disposes of a theorem prover. The most widely available being the language PROLOG, we used its formalism in the rest of the chapter.

16.2 NOTATIONS

In this chapter, we shall use a PROLOG-like notation of the clauses. A Horn clause has the form A:- B, C, D, ... which means, A is TRUE IF (B is TRUE, and C is TRUE, and D is TRUE, ...). A is called the *conclusion* of the clause, and B, C, D, ... are called the *conditions* of the clause. A clause without condition, i.e., a pure positive clause, is often called a *fact*, or a datum. A clause without a conclusion, i.e., a pure negative clause, is called a *question*. It is clear that nothing special is brought by this representation as such. Its interest comes from the fact that an inference mechanism is included in it. For instance, consider the clauses

```
(C1)  MORTAL(x)          :-  HUMAN(x)
(C2)  HUMAN(SOCRATES)    :-
```

In general, one should be able to resolve C_2 and C_1, which means that one should remark that C_2 fulfills the condition of C_1, x being instantiated by SOCRATES. Using PROLOG representation implicitly says that this kind of reasoning, usually called *forward chaining*, will not be used. On the contrary, one will only use *backward chaining*; i.e., a question (represented by a pure negative clause) will have to match some conclusions and will generate new questions etc. ... up to the moment where all questions have been answered. In other words, PROLOG uses a refutation strategy; therefore a question will be stated in the form of a pure negation, as already defined. The original question and all subquestions have been answered when all of them have been put in contradiction with some parts of the data basis. One then says that one has derived the empty clause from the data basis and the question; i.e., that the question is inconsistent with the data basis. For example, the only way to deduce something about Socrates's mortality, is asking the question "Is Socrates mortal?" by adding a pure negative clause in the form

```
(C3)  :-  MORTAL(SOCRATES)
```

The refutation will then proceed by resolving C_3 against C_1, with the substitution [x ← SOCRATES], therefore generating the new question

```
(C4)  :-  HUMAN(SOCRATES)
```

which resolves with C_2 to the empty clause. The refutation procedure thus concludes that it was led to a contradiction by negating MORTAL (SOCRATES), it has therefore proven that asserting it leads to no contradiction.

All that explains why PROLOG programmers say that the proof "succeeds" when the interpreter finds a contradiction, and that the proof "fails" when it does not.

16.3 IMPROVING A GENERALIZATION

In the following, we shall suppose that we reached a given state of knowledge acquisition, and that we are trying to improve it by checking the current state of the learned recognition function against new positive or negative examples. When they do not fit together, the problem is then to be able to improve this recognition function. Said in an intuitive way, one expects from a recognition function to "recognize" new examples and to "reject" new negative examples. Let us now give one possible definition for recognition and rejection. We choose here a quite intuitive way of defining these words. More details have been given in [Kodratoff and Ganascia, 1986; Kodratoff, 1988]. This definition has also been extensively used by J. Nicolas [1986]. The reader is asked to accept these definitions as temporary hypotheses: other definitions would lead to other proofs, but the essential step that we want to illustrate here; for instance, how a proof can be the basis for an explanation, does not depend on these definitions. Let E, NE, and f(x) be an example, a negative example, and a recognition function, respectively.

16.3.1 Recognition of an Example

16.3.1.1 Definitions

Definition 1 One says that a function f(x) *recognizes (1)* an example E when there is an instance of "x," say $[x \leftarrow A]$ such that f(A) = E. This definition fits well our intuitive feeling of "recognition" but, in view of our goal of generating explanations, it is quite inefficient. We shall generalize it somewhat in order to obtain explanation of "why f(x) recognizes E."

Definition 2 Suppose that definition 1 holds; i.e., there is a $[x \leftarrow A]$ such that f(A) = E. Then A is the very instance that exemplifies that both E and $\exists x\ [f(x)]$ are TRUE. Therefore, one says that a function f(x) *recognizes (2)* an example E when there is no contradiction to assert both E and $\exists x\ [f(x)]$ together.

Definition 3 Definition 2 is not very computation efficient, this is why we shall choose to perform the proof by refutation. We shall make the choice to assert E and to refute $\exists x\ [f(x)]$; while the converse, asserting $\exists x\ [f(x)]$ and refuting E could have been also quite possible. Reasons for this choice are discussed at length in Chapter 6 of [Kodratoff, 1988], we shall skip them here. We shall therefore try to prove that asserting both E and $\neg\exists x\ [f(x)]$ together leads to a contradiction. Since $\neg\exists x\ [f(x)]$ is logically equivalent to $\forall x\ [\neg f(x)]$. This gives our third definition. One says that a function f(x) *recognizes (3)* an example E when there is a contradiction to assert both E and $\forall x\ [\neg f(x)]$ together.

This third definition is very computation efficient since E and $\forall x\ [\neg f(x)]$ can be put in clause form, following classical rules [Kowalski, 1979]. Then, if these clauses are Horn clauses, they can be directly given to a PROLOG interpreter,

$$\forall x\ [\neg f(x)]$$

being seen as a question. The proof that E and $\forall x\ [\neg f(x)]$ contradict each other is completed when PROLOG succeeds in finding the empty clause out of their clausal form.

Example 1

Let us suppose that we have so far obtained the following recognition function $f_1(x, y)$ = SPHERE(x) & RED(y) which means: "there are two objects, one is a sphere, the other one is red." Suppose that a further example is given as

$$E_1 = \text{SPHERE(A) \& RED(A)}$$

where "A" is the name of an object that happens to be a red sphere, then E_1 is clearly an instance of $f_1(x, y)$ since the substitution $\sigma_1 = [x \leftarrow A, y \leftarrow A]$ is such that

$$\sigma_1 \circ f_1(x, y) = E_1.$$

Once in clausal form, E reads

```
C1:  SPHERE(A)           :-
C2:  RED(A)              :-
```

and $\forall x\ [\neg f_1(x)]$ becomes

```
C3:                      :-  SPHERE(x), RED(y)
```

Proving that $[\forall x\ [\neg f_1(x)]\ \&\ E_1]$ is contradictory amounts to proving that

$$\{C_1, C_2, C_3\}$$

is contradictory. In order to prove this last statement, one uses resolution [Kowalski, 1979] as follows. SPHERE(x) in C_3 resolves with C_1, with the substitution $[x \leftarrow A]$, leading to the new set of clauses

```
C2:  RED(A)              :-
C'3:                     :-  RED(y)
```

RED(y) in C'_3 resolves with C_2, with the substitution $[y \leftarrow A]$, thus leading to the empty clause. This proves that $\{C_1, C_2, C_3\}$ is contradictory. Therefore, $f_1(x, y)$ recognizes E_1 according to definition 3.

The above example illustrates why we can use the definition of "recognition" we just gave, but it still does not explains its use, since the direct proof by substitution is possible. It may happen that the substitution is very hard or impossible (as Example 2 shows) to find because, in order to make sure that E is an instance of f(x), one must use semantic properties of the microworld one is learning in. In that case, the proof that [$\forall x$ [$\neg f_1(x)$] & E_1] leads to a contradiction may become quite lengthy. An analysis of this proof will show why it succeeds, and explain why f(x) is recognition function for E.

Example 2

Suppose now that the recognition function is $f_2(x)$ = ELLIPSOID(x) & RED(x), which means that we have memorized that "there is a red ellipsoid" in all the scenes we are learning from. Suppose that we are given a further example

E_2 = SPHERE(B) & RED(B)

where "B" is the name of a red sphere. Since ELLIPSOID and SPHERE do not match, the proof by substitution is useless. Suppose now that the semantics of the microworld where learning is taking place are such the following theorem is known

Th_1: $\forall x$ [SPHERE(x) $\Rightarrow$ ELLIPSOID(x)].

We shall add Th_1 in our knowledge base and attempt to prove that

[$\forall x$ [$\neg f_2(x)$ & Th_1] & E_1]

leads to a contradiction. This reads

```
C4:  SPHERE(B)          :-
C5:  RED(B)             :-
C6:  ELLIPSOID(x)       :-  SPHERE(x)
C7:                     :-  ELLIPSOID(x), RED(x)
```

C_7 resolves with C_6, leading to the new clause

```
C8:                     :-  SPHERE(x), RED(x)
```

and C_8 resolves with C_4 and C_5 to lead to the empty clause with the substitution [x $\leftarrow$ B]. Therefore, E_2 is recognized by $f_2(x)$.

As suggested by this example, it must be understood that the proof that

[E & $\forall x$ [$\neg f(x)$]]

leads to the empty clause has to make use of all the available knowledge. Actually, definition 3 tells it implicitly. For the sake of clarity, let us accept some redundancy and introduce explicitly the use of background knowledge in definition 3.

Definition 4 Let BK be the available background knowledge. One says that a function f(x) *recognizes (4)* an example E when there is a contradiction to assert both E and $\forall x$ [$\neg f(x)$ & BK] together. When they are in clausal form, this amounts to say that [E & $\forall x$ [$\neg f(x)$ & BK]] leads to the empty clause.

16.3.2 Rejection of a Negative Example

16.3.2.1 Definitions

As in Section 16.3.1, and for the same reasons, we shall give four definitions of rejection.

Definition 1′ One says that a function f(x) *rejects (1)* an example E when there are no instances of "x," say [x $\leftarrow$ A] such that f(A) = E.

Definition 2′ One says that a function f(x) *rejects (2)* an example E when there is a contradiction to assert both E and $\exists x$ [f(x)] together.

Definition 3′ One says that a function f(x) *rejects (3)* an example E when there is no contradiction to assert both E and $\forall x$ [$\neg f(x)$] together. The proof that E and $\forall x$ [$\neg f(x)$] do not contradict each other is completed when PROLOG is unable to find the empty clause out of their clausal form.

Definition 4′ Let us call BK the background knowledge. One says that a function f(x) *rejects (4)* an example E when there is a no contradiction to assert both E and $\forall x$ [$\neg f(x)$ & BK] together. When they are in clausal form, this is equivalent to saying that [E & $\forall x$ [$\neg f(x)$ & BK]] fails to lead to the empty clause.

Example 3

Suppose that the recognition function is $f_3(x)$ = SPHERE(x) & RED(x) and that NE_3 = SPHERE(C) & RED(D) is a negative example to f_3. Clearly, NE_3 is not an instance of f_3 since x cannot be substituted by both C and D. Using the above formalism, one checks that the system of clauses

```
C9:   SPHERE(C)         :-
C10:  RED(D)            :-
C11:                    :-  SPHERE(x), RED(x)
```

does not lead to the empty clause. Resolving C_{11} with C_9, one obtains

```
C10:  RED(D)            :-
C12:                    :-  RED(C)
```

because "x" has been instantiated by "C" during the resolution. C_{10} and C_{12} cannot be reduced since "C" and "D" are different constants. This proves that NE_3 is a negative example for $f_3(x)$.

All the above illustrates our definitions and the proof procedure but is not very significant as an explanation generator just because there is indeed little to explain. This happens because the amount of background knowledge we are using in these examples is almost zero. However, in Example 2, the proof that

E_2 and $\forall x\ [\neg f_2(x)\ \&\ Th_1]$

contradict each other uses clause

```
C6:  ELLIPSOID(x)          :-   SPHERE(x).
```

Therefore, we can say that $f_2(x)$ *recognizes* E_2 because a sphere is a kind of ellipsoid. This constitutes a simple but not trivial explanation, which happens to be the trace of a formal proof. The next section will show that EBL works in a quite similar way. This will show that both deductive and inductive learning use theorem proving to generate explanations. In the case of inductive learning, the inductive process will generate some recognition function from a set of examples and, once it has been generated by induction, a proving process exactly the same as the one shown in Sections 16.3.1 and 16.3.2 will "explain" why this function recognizes the positive examples and rejects the negative ones it has been learned from.

16.3.3 Improvement of a Formula by an Explanation of Its Success to Recognize a New Example

In cases less toy like than the examples above, it often happens that the recognition function is too much "hairy"; i.e., it contains irrelevant information that does not harm the recognition of the given example, but could be harmful in other situations. One must then prune it from its irrelevant information. This pruning, an essential part of explanation-based learning, will be exemplified below when we shall analyze level 3_3 of the "safe-to-stack" example.

In order to illustrate the necessity for pruning, imagine a recognition function for "man" that contains a predicate "beard" taking the value TRUE when the man has a beard. It may be that only bearded men have been seen so far by the system, which recognizes "man" only if its description gives the value TRUE to "beard."

- Given a complete theory of "man," in this case, a detailed description of what are secondary sexual features, and about some social shaving habits,
- Given a complete description of a bearded man, the system should be able to prove that the predicate "beard" is not necessary to assert that this bearded man is actually a man, because it is a secondary feature often erased by social habits.

As this example shows, since irrelevant information is dropped, another consequence is that some generalization is performed on the given formula. This is the opic of *goal regression* [Waldinger, 1977; Nilsson, 1980], and of explanation-based eneralization (EBG) [Mitchell, *et al.*, 1986].

EBG is very clearly described in [Mitchell, *et al.*, 1986]. We shall insist here on he importance of proof traces in performing it. For that purpose, we shall use a ROLOG version of the "safe-to-stack" example, taken from [Mitchell, *et al.*, 1986].

Such a PROLOG version has been implemented by Kedar-Cabelli [1987]. We have implemented [Touchais, 1987; Siqueira and Puget, 1988; Kodratoff, 1988] a conversational version in which the operationality criterion can be changed at any step by a request of the user. Let us summarize EBG as follows. Given

- a complete theory,
- a training example,
- an operationality criterion given as a list of "operational" predicates. Do the following:
  - prove that the training example fits the theory,
  - gather the corresponding proof tree,
  - prune the proof tree according to the operationality criterion by cutting off any branch containing a predicate that does not belong to the list of operational predicates,
  - generalize the pruned tree while keeping the information sufficient to prove that the training example fits the theory,
  - gather the leaves of the generalized pruned tree: They form a new, operational, description of the general concept described in the training instance.

We shall exemplify this procedure by using the Mitchell, *et al.*, [1986] "safe-to-stack" example. It amounts to learn a more efficient rule to perform stacking. Given

- an example of a particular box "BOX_1 stacked on a particular table "$ENDTABLE_1$."
- a "complete" theory of stacking
- an operationality criterion: Are all predicates met in the theory and the training instance operational?

Find

- an operational rule for stacking objects.

Learning proceeds from three kinds of information. The first kind of information is the "complete" knowledge of the specific box, "BOX_1," and the specific table "$ENDTABLE_1$."

```
C1  ON(BOX1, ENDTABLE1)         :-
C2  COLOR(BOX1, RED)            :-
C3  COLOR(ENDTABLE1, BLUE)      :-
C4  VOLUME(BOX1, 10)            :-
C5  DENSITY(BOX1, 1)            :-
C6  FRAGILE(ENDTABLE1)          :-
C7  OWNER(ENDTABLE1, CLYDE)     :-
C8  OWNER(BOX1, BONNIE)         :-
```

The second kind of information is the "complete" background knowledge about stacking things, also called the theory of the domain.

```
C9  SAFE-TO-STACK(x, y)     :- NOT FRAGILE(y)
C10 SAFE-TO-STACK(x, y)     :- LIGHTER(x, y)
C11 WEIGHT(x, w)            :- VOLUME(x, v),DENSITY(x, d), w is v*d, !
C12 WEIGHT(ENDTABLE1, 50)   :-
C13 LIGHTER(x, y)           :- WEIGHT(x, w1),WEIGHT(y, w2), LESS(w1, w2)
C14 LESS(x, y)              :- x < y
```

The reader may be surprised to find C_{12} placed in the background knowledge. This is due to PROLOG's clumsiness in expressing default values. In new versions of PROLOG that include data typing, it would be possible to declare that "ENDTABLE$_1$" is of type "ENDTABLE" in the above first kind of information and to declare in the second one that the default value of the weight of the type ENDTABLE is 50. Since our aim is not the illustration of typed PROLOGs (that would require a paper by itself), let us go on with a standard PROLOG language and notice that there is no real contradiction to put C_{12} in the background knowledge.

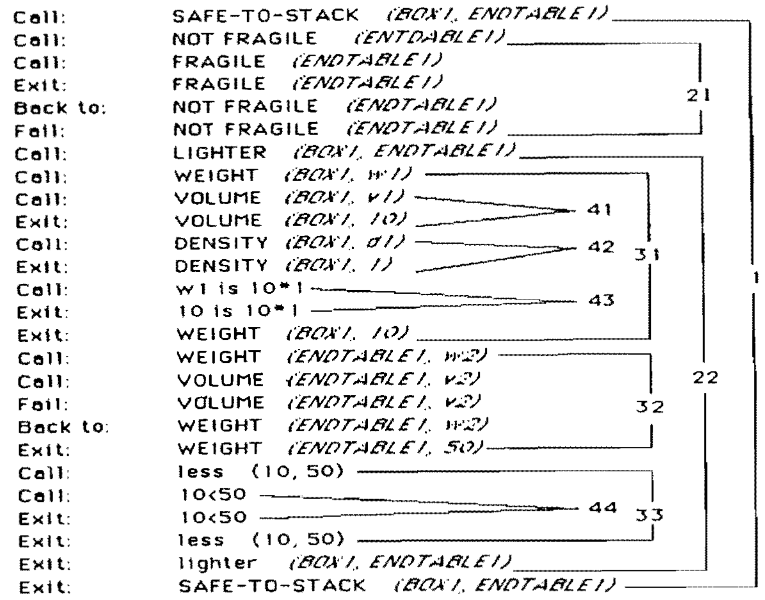
The third kind of information is expressing that one can safely stack "BOX$_1$" on "ENDTABLE$_1$." Since we want to prove that by PROLOG; i.e., by a refutation procedure, this knowledge will be given as question to the PROLOG interpreter. It reads

```
C15         :-  SAFE-TO-STACK(BOX1, ENDTABLE1)
```

Clauses C_9 and C_{10} are supposed to represent a complete knowledge about stacking. Therefore, we already know how to prove or disprove whether an object "x" is stacked on an other "y." What we want to learn from this example is a more efficient (here, this means "more operational") way to stack "x" on "y" than the one given by C_9 and C_{10}.

Most clauses are straightforward rewritings of those in [Mitchell, *et al.*, 1986], but three differences must be pointed out. The first difference is that we use integers; therefore we multiply by 10 the values given in [Mitchell, *et al.*, 1986]. The second difference is that the "ISA" links are dropped. This would have to be expressed as types in a typed PROLOG—as already pointed out, typing PROLOG is not our present topic. The third difference is that the default value is given in clause C_{12} as a "normal" value. The fact that it is a default value is expressed by the "!" at the end of clause C_{11}. If C_{11} succeeds, this "!" tells to "jump over" clause C_{12}. Therefore C_{12} is used only when C_{11} fails as it should be for a default value.

The proof proceeds as the following trace shows. This trace is provided by most PROLOG interpreters. The comment "Call" means that the predicate has been used as a question to the system. The comment "Exit" means that the predicate (with the instances in the "Exit") has been proven TRUE. The comment "Fail" means that the predicate has been proven FALSE. The comment "Back to" means that back-



```
Call:      SAFE-TO-STACK  (BOX1, ENDTABLE1)
Call:      NOT FRAGILE  (ENTDABLE1)
Call:      FRAGILE  (ENDTABLE1)
Exit:      FRAGILE  (ENDTABLE1)
Back to:   NOT FRAGILE  (ENDTABLE1)
Fail:      NOT FRAGILE  (ENDTABLE1)
Call:      LIGHTER  (BOX1, ENDTABLE1)
Call:      WEIGHT  (BOX1, w1)
Call:      VOLUME  (BOX1, v1)
Exit:      VOLUME  (BOX1, 10)
Call:      DENSITY  (BOX1, d1)
Exit:      DENSITY  (BOX1, 1)
Call:      w1 is 10*1
Exit:      10 is 10*1
Exit:      WEIGHT  (BOX1, 10)
Call:      WEIGHT  (ENDTABLE1, w2)
Call:      VOLUME  (ENDTABLE1, v2)
Fail:      VOLUME  (ENDTABLE1, v2)
Back to:   WEIGHT  (ENDTABLE1, w2)
Exit:      WEIGHT  (ENDTABLE1, 50)
Call:      less  (10, 50)
Call:      10<50
Exit:      10<50
Exit:      less  (10, 50)
Exit:      lighter  (BOX1, ENDTABLE1)
Exit:      SAFE-TO-STACK  (BOX1, ENDTABLE1)
```

Figure 16-1: An execution trace directly provided by an interpreter

tracking is taking place. The numbers to the left are those provided by an actual C-PROLOG compiler, we have put on the right the level of embedding they actually represent. For instance, call "NOT FRAGILE(ENDTABLE$_1$)" is indexed by "2" and call LIGHTER(BOX$_1$, ENDTABLE$_1$) is indexed by "4" in the computer output. Actually, they are at the same level of embedding and are labeled respectively 2_1 and 2_2 on the right.

Let us now analyze the above proof trace and show that it actually gives a set of explanations why it is safe to stack BOX$_1$ on ENDTABLE$_1$; which, in the rest of this section will be abbreviated by "expl.stack." A level always begins with a question, labelled as a "Call." When it succeeds, it ends with an "Exit." The exit contains the reason why the call succeeded. This is why we say that each level provides an "expl.stack," that becomes more and more refined as one goes down the levels.

Level 1 is the most outside. In a sense it says

it is safe to stack BOX$_1$ on ENDTABLE$_1$ because I have proven it just now.

It is the most superficial level of explanations. Children use it quite often!

Level 2 contains sublevel 2_1 and sublevel 2_2. Sublevel 2_1 is a failure sublevel: it tells that "NOT FRAGILE" has nothing to do with "expl.stack." We disregard it here, but one must be aware that, when explanations for failures are looked for, then failure sublevels only can provide that information.

Sublevel 2_2 provides the "expl.stack":

It is safe to stack BOX_1 on $ENDTABLE_1$ because BOX_1 is lighter than $ENDTABLE_1$.

Level 3 contains three sublevels 3_1, 3_2, and 3_3. The explanations obtained from each one must be conjuncted to obtain the explanation. They provide the "expl.stack":

It is safe to stack BOX_1 on $ENDTABLE_1$ because the weight of BOX_1 is 10, the weight of $ENDTABLE_1$ is 50, and 10 is less than 50.

One can be tempted to generalize at once by saying that the weight of BOX_1 is w_1, the weight of $ENDTABLE_1$ is w_2, and w_1 is less than w_2. This is not allowed by EBG, which says that one can generalize further only if the numerical values come from example data. If some numerical value is issued from theory data, then this value should be kept as such. In this case, the default value—the weight of $ENDTABLE_1 = 50$—is part of the theory data, not of the example data. Another way to look at this is to say that one must keep them where there is no deeper explanation to the numerical values. In this case, there is no deeper explanation to the fact that the weight of $ENDTABLE_1 = 50$, since sublevel 3_2 contains no inner sublevel. Therefore, this value will be kept in the final result.

Sublevel 3_1 says that the weight of BOX_1 is 10 because its volume is 10, its density is 1, and because $10*1 = 10$. In this case, the numerical values are issued from the example and can be generalized. The value of the volume is called v_1, the value of the density is called d_1, which gives the partial "expl.stack" :

The weight of BOX_1 is $w_1 = v_1*d_1$.

Sublevel 3_3 contains an explanation given by 4_4. This explanation is disregarded because it uses a function, like <, of low level. Deciding what is at "low level" is a strategic decision that must always be made beforehand. In EBG this decision is made by the choice of an *operationality criterion* given by the user, as defined above. In this particular example and choice of operationality criterion, the function < is not an operator that appears in the background knowledge; therefore it does not belong to the list of operational descriptors; therefore we have to prune out level 4_4, which contains it.

Applying this generalization into the explanation of level 3 (which is the last "expl.stack" found) leads to the final "expl.stack":

It is safe to stack BOX_1 on $ENDTABLE_1$" because the weight of BOX_1 is $w_1 = v_1*d_1$, the weight of $ENDTABLE_1$" is 50, and w_1 is less than 50.

The process we describe here, is nothing but a paraphrasing of EBG, with two differences from the original paper [Mitchell, *et al.*, 1986]. First, our presentation has a stronger theorem-proving orientation, similar to the one of [Kedar-Cabelli, 1987]. Second, instead of forcing the variables down to elementary facts, we force the constants up to some level where they can be generalized. In an implementation, EBG is the correct way to realize the transmission of the relations among variables. We felt nevertheless it is easier to understand why this process is an explanation of something when presented the other way round.

DeJong and Mooney [1986] have been presenting a set of criticisms to EBG. We shall not comment here on their criticism except on the one concerning the case where two or more explanations are possible. This point will be more detailed in Section 16.4, since it arises also in the context of explaining the failures. At any rate, as useful as it is, goal regression alone hardly provides the possibility for a progressive improvement of the quality of the explanations. This will be possible only when the theory itself will be improved: Explanations for failures are necessary to improve the theory.

16.3.4 Improvement of a Recognition Function by an Explanation of Its Failure to Recognize a New Example

Let us suppose that f(x) cannot recognize a new example E. As definition 4 says, this means failure to prove that E and $\forall x\ [\neg f(x)\ \&\ BK]$; i.e., one cannot deduce the empty clause from the clause form of $[E\ \&\ \forall x\ [\neg f(x)\ \&\ BK]]$. The problem is now to explain why. This process is usually difficult to implement, as the following example shows.

Example 4

Let $f_4(x)$ = SPHERE(x) & RED(x) and suppose that a new example is E_4: SPHERE(E) & RED(F). One fails to prove that E_4 and $\forall x\ [\neg f(x)]$ contradict each other, as the following set of equivalent clauses shows.

```
C13:  SPHERE(E)        :-
C14:  RED(F)           :-
C15:                   :-  SPHERE(x), RED(x)
```

Remember that in Example 3, NE_3 was considered as a negative example to f_3, therefore failing to prove that it was leading to the empty clause was just fitting definition 4'. Now, since we are considering E_4 as an example, we should have been able to prove it. We must find an explanation for the failure. This is more or less equivalent to finding a new function, $f_4'(x)$, that is "the closest possible" to f_4 but allows the proof to succeed.

The failure can originate from two very different sources. Either there is a problem with the predicates themselves (one cannot find a predicate in the conclusion of the clauses (further called *conclusion-predicate*) to match another one in the conditions of the clauses (further called *condition-predicate*), or there is a problem with the substitutions. Example 4 illustrates the second case. Both condition-predicates SPHERE and RED can match their conclusion counterparts in C_{13} and C_{14}, but "x" must be instantiated either by "E" or by "F."

Example 4'

Imagine that, instead of $f_4(x)$, one would start with

$f_4'(x)$ = SPHERE(x) & RED(y) & RELIGIOUS(z).

Then E_4 would not be recognized by $f_4'(x)$ since the set of clauses

```
C13:  SPHERE(E)         :-
C14:  RED(F)            :-
C15':                   :-   SPHERE(x), RED(y), RELIGIOUS(z)
```

fails to lead to the empty clause. In this case, the variables "x" and "y" could be correctly instantiated by E and F, but no conclusion-predicate could match "RELIGIOUS." Thus E_4 is not recognized by $f_4'(x)$ because E_4 does not contain any information about RELIGIOUS.

When the reason for the failure of the proof is a substitution problem, then one has to introduce variables at the right places to ensure the success of the proof with the new generalization. When the reason for the failure of the proof is a predicate problem, it can be easily found in some cases where one only misses, in the middle of many others that match. When the reason for the failure of the proof mixes substitution and predicate problems, then finding the reason for the failure becomes more or less intractable. This is why we have developed an algorithm described elsewhere [Kodratoff, 1983; Kodratoff, *et al.*, 1985; Kodratoff and Ganascia, 1986; Vrain—Chapter 13, this volume], the role of which is to trace down the possible failures in a given set of examples. The central mechanism for this has been called *structural matching*: It preserves as much as possible the structure of the examples before attempting any generalization.

16.3.5 A Simple Example of Structural Matching (SM)

Consider the two examples in Figure 16–2. Using his intuition, the reader may notice that he can find two different generalizations from these examples. He sees that either

- there are two different objects touching each other, and a small polygon;
- there are two different objects touching each other, one of them is a square.

Both generalizations are true and there is no reason why one of them should be chosen rather than the other. We shall now see that one of the interesting features of SM is that it keeps all the available information, and therefore constructs a formula containing both the above two "concepts."

The examples can be described by the following formulas

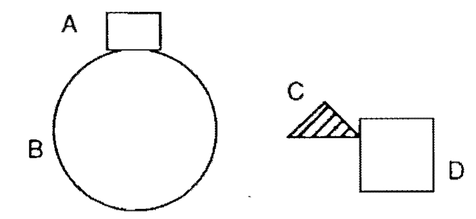


Figure 16-2: Two scenes to be generalized

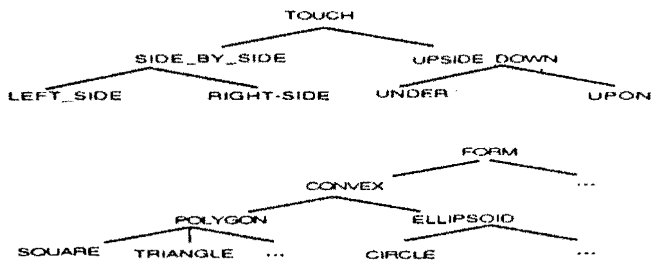


Figure 16-3: A simple hierarchy of geometrical shapes

E_1 = SQUARE(A) & CIRCLE(B) & ON(A, B) & SMALL(A) & BIG(B)
E_2 = TRIANGLE(C) & SQUARE(D) & TOUCH(C, D) & SMALL(C) & BIG(D)

Let us suppose that the hierarchy shown in Figure 16–3 is provided to the system together with the theorems

$$\forall x\, \forall y\, [\mathrm{ON}(x, y) \Rightarrow \mathrm{TOUCH}(x, y)]$$
$$\forall x\, \forall y\, [\mathrm{TOUCH}(x, y) <\Rightarrow [\ \mathrm{TOUCH}(y, x)]$$

This taxonomy and the theorems represent our background knowledge of the semantics of the microworld in which learning is taking place. The SM of E_1 and E_2 proceeds by transforming them into equivalent formulas E_1' and E_2', such that E_1' is equivalent to E_1, and E_2' is equivalent to E_2 in this microworld (i.e., taking into account its semantics). When the process is completed, E_1' and E_2' are made of two parts. One is a variablized version of E_1 and E_2. It is called the body of the SM-ized formulas. When SM succeeds, the bodies of E_1' and E_2' are identical. The other part, called the bindings (of the variables), gives all the conditions at which the bodies might become the examples again. In our example,

Body of E_1' = POLYGON(u, y) & SQUARE(x) & CONVEX(v_1, v_2, z) & ON(y, z) & TOUCH(y, z) & SMALL(y) & BIG(z)
Bindings of E_1' = ((x = y) & (y ≠ z) & (x ≠ z) & (v_1 = ELLIPSOID) & (v_2 = CIRCLE) & (u = SQUARE) & (x = A) & (z = B))
Body of E_2' = POLYGON(u, y) & SQUARE(x) & CONVEX(v_1, v_2, z) & TOUCH(y, z) & SMALL(y) & BIG(z)
Bindings of E_2' = ((x ≠ y) & (y ≠ z) & (x = z) & (v_1 = POLYGON) & (v_2 = SQUARE) & (u = TRIANGLE) & (x = D) & (y = C))

The algorithm that constructs E_1' and E_2' is explained in [Kodratoff, 1983; Kodratoff ınd Ganascia, 1986; Kodratoff, *et al.*, 1985]. The reader can check that E_1' and E_2' ıre equivalent to E_1 and E_2. E_1' and E_2' contain exactly the information extracted from the hierarchy and the theorems which is necessary to put the examples into SM. For instance, in E_1', the expression "(POLYGON(u, y)" means that there is a poly- ;on in E_1, and since we have the binding (u = SQUARE), it says that this polygon is ı square, which is redundant in view of the fact that SQUARE(x) & (x = y) says that ‹ is a square and is the same as y. This redundancy is not artificial when one consid- :rs the polygon in E_2, which is a TRIANGLE.

Once this SM step has been performed, the generalization step becomes trivial: Ve keep in the generalization all the bindings common to the SM-ized formulas and lrop all those not in common.The generalization E_1 and E_2 is therefore

Eg: POLYGON(u, y) & SQUARE(x) & CONVEX(v_1, v_2, z) & TOUCH(y, z) & SMALL(y) & BIG(z) with bindings (y ≠ z).

In "English," this formula means that there are two different objects (named y nd z), y and z touch each other, y is a small polygon, z is a big convex, and there is square (named x), which may be identical to y or z.

The last implementation of SM [Vrain—Chapter 13, this volume] is not tuned › incremental learning. Nevertheless, changes evoked by Vrain would make it work ıcrementally, as long as the structural matching is preserved. An explanation of ıch change due to a new example would then be possible. In other words, it would ɔt be too difficult to include OGUST into an apprentice system as long as the ;ood" structural matching has been found with the first set of examples. In that ıse, the explanations provided by OGUST would be of increasing quality as the ımber of new examples increases. On the contrary, if a new example differs widely om the present generalization, then a completely new generalization process would ıve to take place, thus providing no explanations.

5.3.6 Improvement of a Recognition Function by an Explanation of Its Failure to Reject a New Negative Example

Let us suppose that f(x) recognizes a new negative example NE (as it should ıt). This means that the proof that [NE & ∀x [¬f(x)]] leads to the empty clause

instead of failing as it should if the negative example were rejected. Explaining a success, if not easy, is usually less complicated than explaining a failure. This is why we think that a method can be devised for incremental learning in this case. The simplest way to forbid a proof to succeed is to collect the substitutions that lead to a success and forbid them. Since a substitution can be represented by an equality, the new function will be obtained from the old one, by adding the condition that its variables do not take the values as in the substitution.

Example 5

Suppose that the recognition function is f_5(x, y) = SPHERE(x) & RED(y) and that NE_5: SPHERE(G) & RED(G) is a negative example to f_5. Writing it as clauses, one sees at once that one cannot deduce $\neg f_5$ from NE_5 since

```
C16:  SPHERE(G)        :-
C17:  RED(G)           :-
C18:                   :-  SPHERE(x), RED(y)
```

leads to the empty clause with the substitution [x ← G, y ← G]. Therefore, as defined in Section 16.3.2, NE_5 is recognized by f_5. As one can see the success of the proof is easy to explain: Any substitution that leads to the result is a kind of explanation. The simplest way to change f_5 in order to obtain a new function f_5', that forbids the success of the proof, is to forbid to x and y to take the value G. From f_5 and the new negative example NE_5, one obtains the new recognition function

f_5'(x, y) = SPHERE(x) & RED(y) & ¬[[x = G] & [y = G]].

Let us now see how this can help solve the problems of incremental learning and explanations.

16.3.7 Relative Role of Positive and Negative Examples

Positive examples will generalize to a recognition function containing all the properties common to them. On the other hand, the negative examples express the fact that none of the positive ones may possess such or such property. Therefore, the positive examples must verify the theorem obtained by negating the one that best expresses the properties of the negative examples.

In other words, let us call {E_i} the set of positive examples, and {NE_i} the set of the negative ones. Let NP_j (Negative Property) be a property common to all NE_is. Then, none of the E_is should show NP_j. Let us call f(x) a tentative recognition function of the positive examples, one has to prove that $\forall x\, [f(x) \Rightarrow \neg NP_j]$. This can be done for each j, each proof being necessary to the control of the validity of f(x).

Let us now suppose that we have been able to find $f_N(x)$, the "best" recognition function of all the negative examples. None of the examples should be recognized by this function, then none should be compatible with $\exists x\, [f_N(x)]$, therefore all of them should be compatible with $\neg\exists x\, [f_N(x)] = \forall x\, [\neg f_N(x)]$.

This proves that examples allow us to find recognition functions and that the negative examples allow us to find theorems, like

$\forall x\ [f(x) \Rightarrow \neg NP_j]$ and $\forall x\ [f(x) \Rightarrow \neg f_N(x)]$,

that must be verified by all correct instances of the recognition function. Let then f(x) be a tentative recognition function of the positive examples, one has to prove

$\forall x\ [f(x) \Rightarrow \neg f_N(x)]$

as a condition necessary for f(x) to be a valid recognition function. From this proof one can of course retrieve explanations for why f(x) is a good recognition function.

Note that this formula is valid only for an f(x) that is supposed to be the "best" one; i.e., *the* one that characterizes the domain of the examples. Starting from an f(x), which is not the best one, failure to prove one of the above theorems will help us improve f(x).

In conclusion, negative examples can be used to improve the recognition function. The "best" one must verify $fj\ [\forall x\ [f(x) \Rightarrow \neg NP_j]]$ and $\forall x\ [f(x) \Rightarrow \neg f_N(x)]$.

Example 6

For the remainder of this section, let $f_6(x, y) = SPHERE(x)\ \&\ BLACK(y)$. Suppose that the negative example is

$NE_6 = [BLACK(A)\ \&\ SPHERE(A)]$.

The best theorem is NE_6 itself, the theorem that must be verified by the examples is $\forall x \forall y\ [f_6(x, y) \Rightarrow \neg NE_6]$; i.e., one must attempt to prove

$\forall x \forall y\ [[BLACK(x)\ \&\ SPHERE(y)] \Rightarrow \neg[BLACK(A)\ \&\ SPHERE(A)]]$

and one will, of course, fail.

Let us now see how this progressive refinement of f(x) through failures to prove some $\forall x\ [f(x) \Rightarrow \neg NP_j]$ may be handled. We must attempt to construct a new generalization f'(x) which implies $\neg f_N(x)$. Our method for doing so uses an attempt to prove a particular theorem, henceforth called Th. The reason Th is chosen cannot be understood beforehand; the reader is asked to wait a little before seeing the interesting consequences of its proof.

Th: $\exists x\ [f(x) \Rightarrow f_N(x)]$.

Example 7 (Start)

In this example, we shall attempt to prove that

$\exists x \exists y\ [[BLACK(x)\ \&\ SPHERE(y)] \Rightarrow [BLACK(A)\ \&\ SPHERE(A)]]$

When attempting to prove Th, there are three possible cases.

First Case Th is not provable.

The reason may be that $\forall x\ [f(x) \Rightarrow \neg f_N(x)]$ has been proven. In that case, the new property of the negative examples actually does not cover any example, and nothing has to be changed. It may also be so we are unable to prove both Th and

$\forall x\ [f(x) \Rightarrow \neg f_N(x)]$.

This is the failure case, where nothing can be learned. It shows that we ignore an essential property of the domain, but does not tell where to find it.

Second Case Th is provable and its normal form is the empty clause. We prove it by refutation, attempting to prove that one can deduce the empty clause from ¬Th. Since Th is provable, one will succeed and each success delivers a substitution s, which is an instance of the substitutions to be made to x in order to verify Th. By carrying out all the possible proofs, in the case where there is a finite number of them; or, in infinite cases, by inventing a function that covers all the cases (this part is not emphasized here but is of course very difficult), one defines the set of all the x_i such that $f(x_i) \Rightarrow f_N(x_i)$.

Let us call P_i this set

$P_i = \{x_i / f(x_i) \Rightarrow f_N(x_i)\}$.

We now claim that

$\forall x\ [[f(x)\ \&\ \{x / \neg[x \in P_i]\}] \Rightarrow \neg f_N(x)]$.

This theorem is valid since it simply says that when f(x) is true, and that x belongs to the set for which the implication $f(x) \Rightarrow f_N(x)$ is FALSE, then $f_N(x)$ is FALSE. Though very simple, this theorem gives us the f'(x) we have been looking for:

$f'(x) = [f(x)\ \&\ \{x / \neg[x \in P_i]\}]$.

Example 7 (continued 1)

In order to prove

$\exists\ x\ y\ [[BLACK(x)\ \&\ SPHERE(y)] \Rightarrow [BLACK(A)\ \&\ SPHERE(A)]]$

we shall try to derive the empty clause from its negation ¬Th. As usually [Kowalski, 1979], the theorem is transformed into clauses as follows:

$$\neg\exists x \exists y\ [[BLACK(x)\ \&\ SPHERE(y)] \Rightarrow [BLACK(A)\ \&\ SPHERE(A)]] =$$
$$\neg\exists x \exists y\ [\neg[BLACK(x)\ \&\ SPHERE(y)] \vee [BLACK(A)\ \&\ SPHERE(A)]] =$$
$$\forall x \forall y\ [[BLACK(x)\ \&\ SPHERE(y)]\ \&\ \neg[BLACK(A)\ \&\ SPHERE(A)]]$$

which is equivalent to the set of clauses:

```
BLACK(x)    :-
SPHERE(y)   :-
            :- BLACK(A), SPHERE(A)
```

This set generates once only the empty clause with the substitution

$\sigma = [x \leftarrow A, y \leftarrow A]$.

It follows that the set $\{x = A, y = A\} = P_i$ is the set for which Th is valid. As proven above, conjuncting $\neg[(x = A) \& (y = A)]$ to [BLACK(x) & SPHERE(y)] will yield the looked for generalization

$[\text{BLACK}(x) \ \& \ \text{SPHERE}(y)] \ \& \ [(x \neq A) <186 (y \neq A)]$.

This is the generalization that keeps as much as possible what has been deduced from the examples and excludes the negative example.

One should be aware that this is the best particularization that can be made from the negative examples. If one attempted to derive a more general law, one could overgeneralize and lose some vital information.

Example 7 (continued 2)

Suppose the following further negative example is added

$NE_7 = \text{BLACK}(A) \ \& \ \text{SPHERE}(D)$

hen our method shows that the generalization from f_6 and NE_6 and NE_7 must be

$[\text{BLACK}(x) \ \& \ \text{SPHERE}(y) \ \& \ (x \neq A)]$.

hird Case Th is provable but does not reduce to the empty clause. One must then nalyze the failure. Since we suppose that we use resolution to deduce the empty lause from ¬Th, it follows that the failure will be caused by a subset of irreducible lauses that do not reduce to the empty clause. Let us call LO(x) (Left-Over) this ubset. We shall not give here many details about LO(x), nor comment on the fact at it is not unique in general. Section 16.4 is devoted to an analysis of this kind of roblems.

Let us suppose in this section that it is unique. Consider the expression

$f(x) \ \& \ \neg LO(x)$

d attempt to prove

$Th': \exists x \ [f(x) \ \& \ \neg LO(x) \ ] \Rightarrow f_N(x)]$

Its negation is $\forall x \ [[f(x) \ \& \ \neg LO(x) \ \& \ \neg f_N(x)]$ and, since LO(x) is precisely the ft-over of the resolution of $\forall x \ [f(x) \ \& \ \neg f_N(x)]$, the empty clause will always follow om it. Let us now call P_i the set of values that verify Th′.

Drawing a conclusion from the above reasoning requires us to take a somewhat oser look at LO. Due to the conjunctive form of the theorem

$\forall x \ [f(x) \ \& \ \neg LO(x) \ \& \ \neg f_N(x)]$,

one can always assume that each clause contains atoms that originate from and only from f(x) or from and only from $f_N(x)$. It follows that LO(x) has the form

$LO'(x) \ \& \ \neg LO''(x)$ where $f(x) = f'(x) \ \& \ LO'(x)$ and $f_N(x) = f_N'(X) \ \& \ LO''(x)$.

During the proof of Th′, LO′ (respectively ¬LO″) resolves some predicates of $\neg f_N$ (respectively f). It follows that from the proofs of $\exists x \ [[f(x) \ \& \ \neg LO(x) \ ] \Rightarrow f_N(x)]$ one can deduce that

$\forall x \ [[f'(x) \ \& \ \{x/\neg[x \in P_i]\}] \Rightarrow \neg f_N'(x)]$

by the same reasoning as above.

Let us now use the two following trivialities

from $A \Rightarrow C$, deduce that $A \ \& \ B \Rightarrow C$
from $A \Rightarrow C$, deduce that $A \ \& \ D \Rightarrow C \vee D$

in order to find that

$\forall x \ [[f(x) \ \& \ \{x/\neg[x \in P_i]\} \ \& \ \neg LO''(x)] \Rightarrow \neg f_N(x) \ ]$

which is the "interesting" form we wanted to construct.

The final definition we can now give for the best formula that can be learned from formula f(x) and negative examples generalizing to $f_N(x)$ is

$[\ f(x) \ \& \ \neg LO''(x) \ \& \ \{x/\neg[x \in P_i]\}]$.

This is the correct recognition function.

We now describe two simple examples showing that our definitions contain the well-known intuitive learning behavior when the positive and negative examples mismatch by a predicate. There are two cases.

First Case The generalization from examples contains more predicates than the negative example.

Example 7 (continued 3)

Recall that $f_6(x, y)$ is: BLACK(x) & SPHERE(y), and suppose that the negative example is now BLACK(A). The attempt to prove

$\exists x \ y \ [[\text{BLACK}(x) \ \& \ \text{SPHERE}(y)] \Rightarrow \text{BLACK}(A)]$

fails with SPHERE(y) as LO = LO′. Conjuncting ¬LO to $f_6(x, y)$, one attempts now to prove

$\exists x \exists y \ [[\text{BLACK}(x) \ \& \ \text{SPHERE}(y) \ \& \ \neg \text{SPHERE}(y)] \Rightarrow \text{BLACK}(A)]$,

i.e., that $\neg\exists x \exists y \ [\ \text{BLACK}(x) \Rightarrow \text{BLACK}(A) \ ]$ contains a contradiction. The substitution $[x \leftarrow A]$ describes the domain where this contradiction holds, and it follows that

$$\forall x \forall y \ [[\text{BLACK}(x) \ \& \ \text{SPHERE}(y) \ \& \ \neg\text{SPHERE}(y) \ \& \ (x \neq A)] \Rightarrow \neg\text{BLACK}(A)].$$

This shows that the final generalization is

$$[BLACK(x) \& SPHERE(y) \& SPHERE(y) \& (x \neq A)] = [BLACK(x) \& SPHERE(y) \& (x \neq A)]$$

Second case The generalization from examples contains fewer predicates than the negative example.

Example 8

Let the generalization from examples be $f_7(x) = BLACK(x)$, and the negative example be SPHERE(A) & BLACK(A). The left-over is $\neg SPHERE(A) = \neg LO''$, conjuncting its negation to BLACK(x) allows us to find the empty clause with

$[x \leftarrow A]$.

It follows that the best recognition function is

$BLACK(x) \& \neg SPHERE(A) \& (x \neq A)$.

16.4 IMPROVING THE EXPLANATIONS

16.4.1 Improving the Quality of the Generalization

Instead of applying the above techniques to the recognition function and a negative example, one can also attempt to compare it to a generalization of the negative examples. Since their generalization will be used in order to expel some information from the recognition function of the examples, it may be that the modified recognition function no longer recognizes all the examples after its modification. This will happen when one uses an overgeneralization of the negative examples. It is therefore extremely important to avoid this overgeneralization. Structural matching, the role of which is to avoid such kind of overgeneralization, is important when generalizing examples, but it is even more important when generalizing negative examples.

Example 9

Consider again $f_6 = BLACK(x) \& SPHERE(y)$.Consider now the case where one wants to find the correct generalization associated to f_6 and

$NE_8 = BLACK(A) \& SPHERE(A)$
$NE_9 = BLACK(B) \& SPHERE(B)$.

The recognition function deduced from NE_8 and NE_9 is

$f_N(x) = BLACK(x) \& SPHERE(x)$

In this case, Th is

$\exists x \exists y\ [[BLACK(x) \& SPHERE(y)] \Rightarrow [BLACK(x) \& SPHERE(x)]]$

which is TRUE for the unique substitution $[y \leftarrow x]$, therefore P_i is characterized by $x = y$ and $\neg[x \in P_i]$ if $x \neq y$. It follows that the correct generalization is, in this case,

$[BLACK(x) \& SPHERE(y) \& (x \neq y)]$.

Notice that overgeneralizing NE_1 and NE_3 to BLACK(x) & SPHERE(y) for instance would lead to total disappearance of the recognition function—a case clearly difficult to overcome by further modifications!

16.4.2 Improving of the Quality of the Proof

As seen in preceding sections, an explanation procedure can always be attached to a proof of recognition or rejection. There are often several possible proofs. Each of them will provide new different explanations. Explanations relative to successes will enrich the recognition functions. Explanations relative to failures will allow modifying the data basis. For instance, in the example of Section 16.3.3, the default values of an ENDTABLE can be modified in case the system fails to recognize that a given BOX_1 can be stacked on a given $ENDTABLE_1$. In a real situation, where we will have to handle numerous explanations, one will be able to modify the data theory by learning.

16.4.2.1 The System Generates Several Explanations from One Example

Suppose that, as in Section 16.3.3, it finds that BOX_1 can be stacked on $ENDTABLE_1$ because the weight of BOX_1 is less than 50. Suppose also that, by using another reasoning path, it finds also the other explanation: because $ENDTABLE_1$ is "very stiff" (one should have defined this predicate in the data of the theory). The system will have first to prove that there is no mutual implication between "BOX_1 is less than 50" and "$ENDTABLE_1$ is very stiff." Then it will have to improve its explanation by providing a disjunction of these two cases. As another example, suppose that one obtains the two explanations

Ex_1: the weight of BOX_1 is less than 50
Ex_2: the weight of $ENDTABLE_1$ is more than twice the weight of BOX_1

depending on the reasoning path that is used during the proof. In this case, one would have to check that the weight of BOX_1 is less than 25; i.e., that the two explanations are not contradicting each other. None of them imply the other, they would have both to be kept.

16.4.2.2 The System Generates Several Explanations from Several Examples

We shall present here the ideas underlying a system currently under construction [Duval and Kodratoff, 1990] in our French group. This system performs EBL on a set of several training examples and combines the explanations. Let us illustrate our approach on the "suicide" example of DeJong and Mooney [1986]. Suppose that a first example describes the suicide of Peter with a gun. For a given operationality criterion and given schemes for "killing," let us suppose that EBL generalizes this instance to "a man self-killing with a shot-gun." Suppose that we get now the story of Mary's suicide with a gun and that we generalize it to "a woman self-killing with a shot-gun." Using a standard learning procedure (here, by using the "climbing the generalization tree" rule), we can combine the two above generalizations into the unique one "a human self-killing with a shot-gun." The same kind of mechanisms, now including structural matching or even some analogy-based reasoning [Vrain and Kodratoff, 1987; 1988] can be used when getting new examples of suicide with, say, a dagger or with butane gas (shot-gun would then have to be generalized to "harmful object"). A more difficult case will be met when the system will see a suicide committed by jumping from the Eiffel tower, which is not normally a harmful object. The learning would then proceed by recognizing the analogies between "pushing someone from a cliff" and "jumping from the Eiffel tower." The whole "killing" scheme would then have to be refined by including the situations where an actor place an other one in a potentially harmful situation etc. ...

This example shows that we need a system able to refine the theory it starts with; i.e., that includes new rules into the theory, in order to avoid proof failures. The next section describes some of the ideas underlying such a system.

16.4.3 Refinement of Incomplete Theories

A stringent requirement of EBL is that it demands a complete theory of the domain of learning. One can easily imagine that this case seldom holds in reality. This is why a great deal of work has been done on the topic of error recovery by explanations, which is quite similar to explanation improvement. The interested reader should consult the chapters in this volume by Carbonell and Gil, DeJong and Mooney, Hirsh, and also [Hill, 1987; Rajamoney and DeJong, 1987; Mostow and Bhatnagar, 1987].

For the management of incomplete theories, the central idea is to complete the failure proofs by an abduction[1] mechanism guided by analogical reasoning about explanations and enables to discover new rules that refine the domain theory. We shall illustrate this by the following example drawn from DeJong and Mooney [1986]. In this example, our aim is learning a definition of the concept of suicide Kill(x,x). The domain theory TH_0 contains the following rules

[1] Technically, an abduction is the process that, given a B, finds some A's such that $A \Rightarrow B$. In practice, this amounts to finding the conditions that would allow a failed proof to succeed.

```
KILL(a,b)        :-   HATE(a,b), POSSESS(a,c),
                      WEAPON(c)
HATE(w,w)        :-   DEPRESSED(w)
POSSESS(u,v)     :-   BUY(u,v)
WEAPON(z)        :-   SHOT-GUN(z)
```

where a, b, c, w, u, v, z are variables.

The training instance TR1 is a suicide, described by the following facts.

```
TR1
DEPRESSED(JOHN)    :-
BUY(JOHN,OBJ1)     :-
SHOT-GUN(OBJ1)     :-
                   :-   KILL(JOHN, JOHN)
```

DeJong and Mooney have shown that applying EBL to the training instance leads to building the generalized proof of the fact that John has been committing suicide, that we shall from now on call S (Source) in which x and c are variables (see Figure 16–4).

This produces the rule

```
KILL(x,x)  :-              DEPRESSED(x), BUY(x,c), SHOT-GUN(c)
```

which is "added" to TH_0 in order to define the concept C of suicide. Nothing is actually added since this new rule is a consequence of TH_0. It happens that this rule seems to be more operational to define suicide that applying KILL(a, b) with a = b.

Suppose now that the system is provided with an example TR2 of concept C, which is not recognized by the theory. In other words, some oracle will have been telling the system that TR2 does belong to the suicide case, but the PROLOG solver failed to prove it. Nevertheless, it may happen (this is the case we are actually considering here) that partial proofs might be obtained. They will provide partial proof trees that may match parts of S. Let us call Ti (Target incomplete) incomplete proof trees that can be viewed as partial explanations as explained with more details in [Duval and Kodratoff, 1990]. Our aim will then be to complete each of these Ti into Tc (Target complete) in such a way that S and Tc are as "close" as possible.

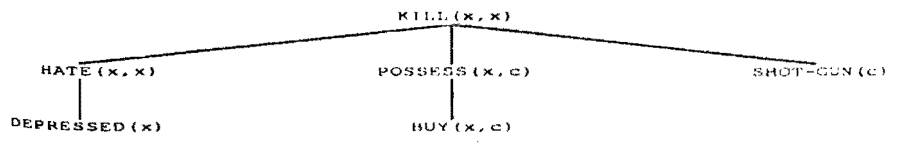


Figure 16-4: Generalized proof that John has been committing suicide

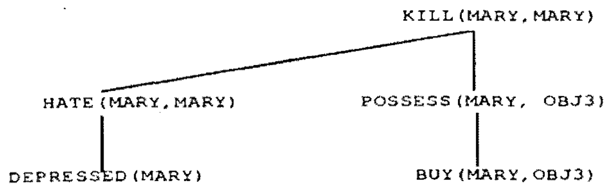


Figure 16–5: Ti: Partial explanation of Mary's suicide

As an illustration of such a Ti, consider an other suicide instance TR2 described by the following facts.

```
DEPRESSED(MARY)             :-
BUY(MARY,OBJ1)              :-
SLEEPING-PILLS(OBJ1)        :-
PRICE(SLEEPING-PILLS, 6)    :-
BUY(MARY, OBJ2)             :-
BOOK(OBJ2)                  :-
PRICE(BOOK, 5)              :-
...
```

where OBJ1 and OBJ2 are constants.

We will be unable to prove KILL(MARY, MARY) because she has no shotgun. Nevertheless we obtain one, and only one in this case, partial proof (see Figure 16–5).

Ti obviously matches a subtree of S.

In order to achieve the completion of Ti into Tc, and to achieve it so that Tc is "close" to S, we propose to use two different abductions. In general, the mechanism underlying this abduction process is analogy. We shall not speak of analogy here—see [Duval and Kodratoff, 1990] for more details.

First Abduction We attempt to complete Ti by viewing it as an instance of S (this is one of the possible definitions of "closeness"). Therefore, Ti will be completed by taking the missing pieces from S, appropriately instantiated.

In our example, such a forced matching leads to Tc1 (see Figure 16–6).

This first abduction can be seen as interpreting the fact that Mary possesses an object that is in fact a shot-gun, but our knowledge about shot-guns is not sufficient to identify this object. The cause of the preceding failure is attributed to our supposed "ignorance" that OBJ3 (i.e., sleeping pills) are actually shot-guns. Notice that this first mechanism is the only one considered in other works about abduction such as [Cox, 1986], and note how easily it can lead to absurd results, as here.

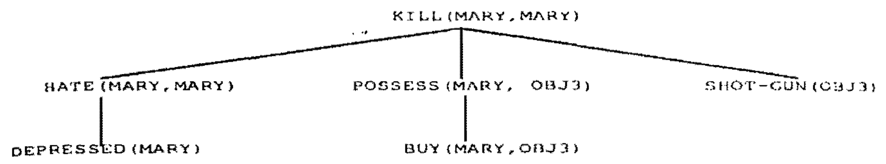


Figure 16–6: Tc1: Proof tree completed. S matches Tc1 with the substitutions [x ← MARY, c ← OBJ3].

Second Abduction In this case, we try to add a new rule that will allow us to complete the proof. One can easily understand why this mechanism has not been taken into account so far; in principle one can add so many ridiculous rules that this approach seems to be hopeless.

For instance, in our example adding to TH_0 the rule

```
KILL(x,x)   :-  PRICE(SLEEPING-PILLS, 6)
```

will indeed allow us to prove Mary's suicide. But it means that everyone will commit suicide when the price of sleeping pills reaches the value 6, which is totally irrelevant to the preceding suicide case, notwithstanding its strange meaning.

In order to avoid adding such ridiculous rules as above, we define a new notion of distance between S and Tc. Given two possible completed proof trees, Tc and Tc′, consider an attempt to match S and Tc, and S and Tc′. In this second abduction, we do not consider the case where they match (this would be the first kind of abduction). We shall therefore collect the mismatches between S and Tc on the one hand, and between S and Tc′ on the other hand. We shall say that that Tc is closer to S than Tc if the number of mismatches between Tc and S is less than the number of mismatches between Tc′ and S. When they are equal, we shall say that Tc is closer to S than Tc′ when the conceptual distance (supposedly defined) between the mismatches is less for Tc than for Tc′.

In our example, it is clear that the number of mismatches between S and the proof tree obtained by using

```
KILL(x,x)   :-  PRICE(SLEEPING-PILLS, 6)
```

to prove Mary's suicide is very high.

On the other hand, completing Ti by SLEEPING-PILL(OBJ3), obtaining thus the proof tree Tc2, shown in Figure 16–7, leads to one mismatch only, namely

[SHOT-GUN(OBJ3) ← SLEEPING-PILLS(OBJ3)].

Notice that we speak here of mismatch since SHOT-GUN(OBJ3) is not a variable. Applying EBG to this proof leads to add the new rule

```
KILL(x,x)   :-  DEPRESSED(x), BUY(x,c), SLEEPING-PILLS(c)
```

to TH_0.

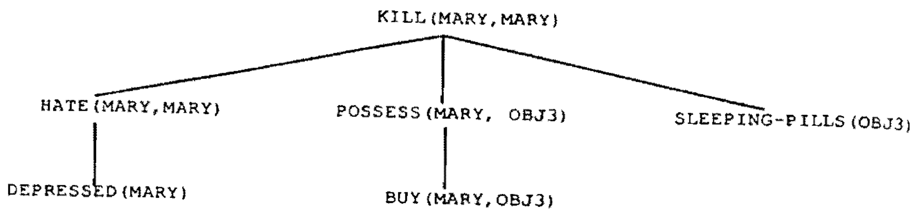


Figure 16–7: Sleeping pills are viewed as the cause of Mary's death

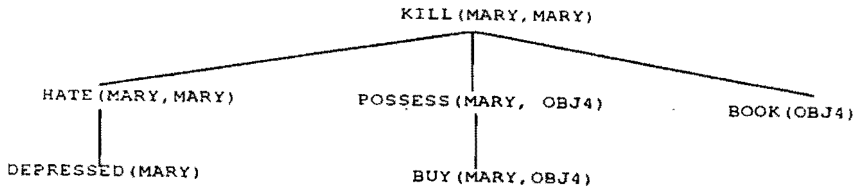


Figure 16–8: Books are viewed as the cause of Mary's death

One can complete it as well by BOOK(OBJ4), obtaining the proof tree Tc3 shown in Figure 16–8.

Applying EBG to this proof leads to add the new rule

```
KILL(x,x)   :-   DEPRESSED(x), BUY(x,c), BOOK(c)
```

to TH_0.

From our semantic knowledge, it is clear that Tc2 is the good answer, but this result is not so easy to obtain. It needs to have some ways to prove that sleeping pills are more likely to kill someone than books.[2] Proving this would ask a large amount of extra-knowledge not contained in TH_0. Anyhow, we simply suppose here that one has been able to set up some way of evaluating the semantic distance between SHOT-GUN and SLEEPING-PILLS, and between SHOT-GUN and BOOK, and that the first is less than the second. Then, we would choose Tc3 as other possible abduction.

We will leave here open the two following points. One is the way to measure the semantic distance between concepts, which can be taken care of completely independently of this chapter. The second one is to choose between the results of the above two kinds of abductions. A a simple hint, let us point out that the new rule

[2] Consider however that a French writer recently published a book entitled *A Cheap Guide to Happy Suicide*. He is regularly sued by families of suicides arguing that his book is the cause of the suicide's death, and the families always win their cases.

should be checked against other knowledge of different granularity, for instance finding if shot-guns and sleeping pills generalize to a rich concept or not.

As conclusion to this section, let us acknowledge first that the difficult problems of completing the theory from experiences and explanation are visibly very hard to tackle with. Nevertheless, we can see that proposing sound abduction is quite feasible, and that choosing the best among them is a matter of good structuration of the background knowledge.

16.5 CONCLUSIONS

This chapter advocates an integrated approach to machine learning in which inductive and deductive learning enhance each other. The key concept to link them is *explanations*, and the source of explanations is a theorem prover. We have shown how much EBL relies on the trace of proof, and how much the construction of a recognition function relies as well on theorem proving. EBL uses a proof that some goal may be achieved, like the goal of stacking BOX_1 on $ENDTABLE_1$ as shown in Section 16.3.3. We have shown that learning from examples needs to prove or disprove that an instance belongs to a concept, in exactly the same way as EBL does.

In other words, this chapter can be considered a detailed illustration of how constructive learning is going to be achieved. One may have to learn rules to solve a problem, to achieve a goal, to recognize a concept but, in all cases, constructive learning will proceed as follows.

1. Choose a representation language for the examples of concept or behavior.
2. Find a generalization that will describe some of the properties they share.
3. Prove, or disprove, that other examples are instances of the generalization obtained at step 2.
4. Analyze the proof, or proof failures, in order to obtain an explanation for why the new example is recognized or rejected.
5. Use these explanations to improve the representation language of the examples (i.e., use new explanations to perform again, and better, step 1 above) and their generalization (i.e., use new explanations to perform again, and better, step 2 above).

In the description of this closed-loop process, the first four steps have already received much attention in the current literature. Step 5 has to be somehow implemented in all the systems described in the *Integrated Learning* part of this book. We have shown here that the whole integration hinges on explanations. Our main argument is that being able to refine explanations will, in turn, allow us to refine the generalizations. We did not argue here about the representation changes, but we are convinced that explanations of failures are the best way to drive the search in the best direction whenever representation changes are needed.

References

val, B. and Kodratoff, Y. 1990. "A Tool for the Management of Incomplete Theories: Reasoning about Explanations," in *Machine Learning, Meta-Reasoning and Logics*, P. Brazdil and K. Konolige (eds), Kluwer Academic Press, Amsterdam, pp. 135–158.

l, W. L. 1987. "Machine Learning for Software Reuse," *Proc. IJCAI-87*, pp. 338–344, John McDermott (ed), Morgan Kaufmann, San Mateo, CA.

lar-Cabelli, S.T. and McCarthy, L.T. 1987. "Explanation-Based Generalization as Resolution Theorem Proving," *Proc. 4th International Workshop on Machine Learning*, pp. 383–389, Irvine, June 1987, Langley, P. (ed), Morgan Kaufmann, San Mateo, CA.

lratoff, Y. 1983. "Generalizing and Particularizing as the Techniques of Learning," *Computers and Artificial Intelligence* 2, pp. 417–441.

lratoff, Y., Ganascia, J.-G., Clavieras, B., Bollinger, T., and Tecuci, G. 1984. "Careful generalization for concept learning," in *Advances in Artificial Intelligence*, T. O'Shea (ed), pp. 229–238, North-Holland Amsterdam, 1985.

lratoff, Y. and Ganascia, J.-G. 1986. "Improving the Generalization Step in Learning," in *Machine Learning: An Artificial Intelligence Approach, Volume II*, Michalski, R.S., Carbonell, J.G., and Mitchell, T.M. (eds), Morgan-Kaufmann, San Mateo, CA, pp. 215–244.

lratoff, Y. and Tecuci, G. 1987a. "DISCIPLE-1: Interactive Apprentice System in Weak Theory Field," McDermott, J. (ed), *IJCAI 1987*, Morgan Kaufmann, San Mateo, CA, pp. 271–273.

—— , 1987b. Techniques of design and DISCIPLE learning apprentice. *International Journal of Expert Systems*, 1, pp. 39–66.

lratoff, Y. 1988. *Introduction to Machine Learning*, Pitman, London. Available in the U.S. from Morgan Kaufmann, San Mateo, CA.

valski, R. 1979. *Logic for Problem Solving*, North Holland.

halski, R.S. 1983. "A theory and a methodology of inductive learning." *Artificial Intelligence*, 20, pp. 111–161.

—— , 1984. "Inductive learning as rule-guided transformation of symbolic descriptions: A theory and implementation," in *Automatic Program Construction Techniques*, Biermann, A.W., Guiho, G., Kodratoff, Y. (eds), Macmillan Publishing Company, pp. 517–552.

Michalski, R.S. and Stepp, R.E. 1983. "Learning from Observation: Conceptual Clustering," in *Machine Learning: An Artificial Intelligence Approach*, Michalski, R.S., Carbonell, J.G., Mitchell, T.M. (eds), Morgan Kaufmann, San Mateo, CA, pp. 163–190.

Mitchell, T.M. 1983. "Learning and Problem Solving" *Proc. IJCAI-83*, pp. 1139–1151, Karlsruhe.

Mitchell, T.M., Mahadevan, S., and Steinberg, L.I. 1985. "Leap: A Learning Apprentice for VLSI Design," *Proc. IJCAI-85*, Los Angeles, pp. 573–580.

Mitchell, T.M., Keller, R.M., and Kedar-Cabelli, S.T. 1986. "Explanation-Based Generalization: A Unifying View," *Machine Learning* 1, pp. 47–80.

Mostow, J. and Bhatnagar, N. 1987. "Failsafe—A Floor Planner that Uses EBG to Learn from its Failures," *Proc. IJCAI-87*, McDermott, J. (ed), pp. 249–255, Morgan Kaufmann, San Mateo, CA.

Nicolas, J. 1986. "Learning as Search: A Logical Approach," *CIIAM'86*, Marseille, Hermes, Paris, pp. 441–459.

Nilsson, N. J. 1980. *Principles of Artificial Intelligence*, Morgan Kaufmann, San Mateo, CA.

Rajamoney, S. and DeJong, G. 1987. "The Classification, Detection and Handling of Imperfect Theory Problems," *Proc. IJCAI-87*, McDermott, J. (ed), Morgan Kaufmann, San Mateo, CA, pp. 205–207.

Siqueira, J.L. and Puget, J.F. 1988. "Explanation-Based Generalization of Failures," *Proc. ECAI*, Munich, Kodratoff, Y. (ed), Pitman, London.

Touchais, R. 1987. "Généralisation à partir de l'explication," Rapport de stage DEA, Orsay Univ.

Vrain, C. and Kodratoff, Y. 1989. "The Use of Analogy in Incremental SBL," in *Knowledge Representation and Organization in Machine Learning*, Lecture Notes in Computer Science, No. 347, Morik, K. (ed), Springer-Verlag, Berlin, pp. 231–246.

Waldinger, R. 1977. "Achieving Several Goals Simultaneously," in *Machine Intelligence* 8, Elcock, E.W. and Michie, D. (eds), Ellis Horwood.

COMMENTARY

Robert E. Stepp
(University of Illinois)

1 OVERVIEW OF THE PAPER

Kodratoff describes an approach to machine learning that performs in a manner he calls an "explicatory apprentice"; i.e., a system designed for building and improving explanations of events. The paper focuses on theorem-proving techniques that bring together the two polar notions in machines learning: similarity-based learning (SBL) and explanation-based learning (EBL). It is pointed out that these approaches are actually closely related—both aim at producing good explanations of events, and both can be practiced using theorem-proving techniques (although doing so naturally introduces powerful preferences and biases, a topic not considered).

Kodratoff notes that SBL and EBL operate in problem-dependent microworlds. In traditional SBL approaches, the microworld is the background knowledge provided to the learning system. In traditional EBL approaches, the microworld is the domain theory by which the system understands the concept being explained.

First, an explanation can be viewed as a *generalized proof*. In EBL, explanation as logical proof provides the constraints that must be maintained for the explanation to hold. It is this proof mechanism that requires EBL applications to include complete domain knowledge. In SBL systems, explanation is the mechanism of performance. By the explanation, SBL systems can ultimately identify the categories to which events should be assigned, and thereby explain them.

Second, an explanation can be viewed as a learning/generalizing tool. In EBL, explanation maintenance is the tool by which the boundary between generalization and overgeneralization is defined. EBL works because generalization is never taken to the point that the explanation ceases to hold. In SBL, generalization (and inverse generalization, i.e., specialization) is a mechanism for explanation repair. New events that do not receive the desired decision (those that are misidentified) call for explanation repair.

In the language of covering algorithms [Michalski, 1984], this situation is described as finding one or more events that are either uncovered or covered by the

wrong category description. Kodratoff presents a theorem-proving approach involving cases where an explanation proof fails when it should succeed or succeeds when it should fail. In both cases, repair is required.

From a theorem-proving standpoint, the mechanisms of SBL and EBL are related. For EBL systems, the main inference rule is *modus ponens*:

if A *and* A $\Rightarrow$ B *then deduce* B.

For SBL systems, Kodratoff says the main inference rule is *classical generalization*:

if A *and* A $\rhd$ B *then deduce* B

where the symbol $\rhd$ means "is more general than." In each case, the mechanisms are to be used to generate proofs (and generalized proofs) called *explanations*, which are in the language of the problem domain. Guidance on how to do this is embodied in the preferences and biases of the algorithms, and in "explanation kits" provided by domain knowledge. In EBL, the explanation kit is the knowledge needed to do goal regression. In SBL, the explanation kit is a forest of concept generalization trees to climb (e.g., squares, rectangles, triangles are generalize to the concept *polygon*).

2 THE SBL VIEW

In this section of the commentary, the link between Kodratoff's approach and Michalski's approach is explored. Michalski has presented the formula (see Chapter 3, this volume):

Hypotheses + Background-Knowledge $\rhd$ Facts

By this statement he means that the hypotheses and the background knowledge taken together account for or *explain* the facts. If one makes the background knowledge implicit rather than explicit (by operating in the *context* of the background knowledge), and if one calls the hypotheses the *recognition rules*, then one can arrive at this commentator's interpretation of the view taken by Kodratoff, illustrated in the following diagram in which the box encloses the context within which the background knowledge is available to be applied.

$$\boxed{\begin{array}{l}\text{Recognition rules} \rhd \text{Examples}\\ \text{Background Knowledge}\end{array}}$$

Thus, Kodratoff's SBL approach can be described by Michalski's formula; the goal is the generation of recognition rules. An EBL approach is described by the same formula; the goal is to add to the background knowledge an improved/generalized explanation proof.

Kodratoff's approach is to develop a system to manage explanations. Learning is by explanation repair. To do this, one must have at least four basic knowledge management tools:

- a generalization operator
- a specialization operator
- a consistency maintenance mechanism
- preference criteria

3 AN ILLUSTRATION AND BRIEF REVIEW OF EXPLANATION REPAIR

According to Kodratoff, a learning system needs to perform at least three tasks:

- learn to recognize positive examples,
- learn to reject negative examples, and
- manage consistency

Here are a few thoughts about each, drawn from the paper.

3.1 Learning to Recognize Positive Examples

Suppose there is a rule (a recognizer) of the form

R_1: $\exists$ x y (sphere x) (red y)
"There exists a part x that is a sphere and a part y that is red"

and there are examples such as

E_1: (sphere A) (red B)	{covered}
E_2: (sphere A) (red A)	{covered}
E_3: (rectangle A) (red B)	{not covered}

Examples E_1 and E_2 are properly explained (covered) by R_1. Example E_3 causes R_1 to fail. The system needs to generalize on failure to repair its recognizer and explain all the examples, based on the fact that the system cannot explain how E_3 is an instance of R_1.

From the microworld (background knowledge) the following useful inferences are retrieved.

$\forall$ x (sphere x) $\Rightarrow$ (ellipsoid sphere x)
$\forall$ x (ellipsoid sphere x) $\Rightarrow$ (convex ellipsoid sphere x)
$\forall$ x (rectangle x) $\Rightarrow$ (polygon rectangle x)
$\forall$ x (polygon rectangle x) $\Rightarrow$ (convex polygon rectangle x)

By applying these inference rules to the examples, discarding unlike forms, and changing constants into variables within the like forms, a process Kodratoff calls *structural mapping* produces the form

R_2: $\exists$ v_1 v_2 x y (convex v_1 v_2 x) (red y)

R_2 is the repaired version of R_1 and recognizes all examples *when operated in the context of the background knowledge*. This rule is interpreted as "there is a convex part and a red part." *Structural mapping* is the process for making the minimum (most specific) generalization using the transformations of climbing a specificity hierarchy and variablizing forms. Climbing the specificity hierarchy uses deductive inference over implications that define the hierarchy. It is thus one type of knowledge-based generalization transformation. Vrain [Chapter 13, this volume] describes another knowledge-based generalization mechanism.

Structural mapping makes inductive leaps (when discarding unlike forms and doing variablizing) and is an SBL mechanism. Not explained in Kodratoff's paper is how structural mapping can limit its generalization to avoid describing negative events (and how to do this efficiently). Kodratoff provides references to papers that do investigate this [Ganascia, 1985; Vrain, 1985]. Since the resulting rules are already maximally specific under this paradigm, the introduction of negative events could lead to a disjunctive statement or (if disjunction is not allowed) to only a partial explanation of a subset of the examples.

3.2 Learning to Reject Negative Examples

Suppose we have arrived at rule R_2, and we encounter negative example E_4:

E_4: (sphere A) (red A) (rectangle B)

E_4 is recognized by R_2, and this is incorrect. Here specialization is driven by explanation failure. A proof of R_2 given the examples shows that E_4 supports R_2 under two bindings of variables, namely (x/A, y/A) and (x/B y/A). To repair the rule, both bindings must be defeated. This is done using specialized binding and idempotency to get

R_3: $\exists$ v_1 v_2 v_3 v_4 x y z (convex v_1 v_2 x) (convex v_3 v_4 z) (red y)
([x$\neq$A] v [y$\neq$A]) ([x$\neq$B][y$\neq$A])

This appears to be a rather awkward notation. The approach is similar to Winston's [1975] introduction of *must not* conditions in his blocks-world learning algorithm. Learning a generalized constraint on bindings requires considering several negative examples. Kodratoff proposes that constraints on bindings, such as in R_3 above, be generalized by requiring conjunctive forms that are maximally specific. This eliminates the full enumeration of all binding constraint combination without additionally restricting the concept.

3.3 Managing Consistency

Given a rule R that explains positive examples, a rule R_c is consistent with R if R can be shown to be more specific than (not R_c). Let R involve a set of variables denoted as *vars* and let R_c involve a set of variables denoted as *cvars*. Consistency could be demonstrated in one of the three ways:

$\forall$ vars R $\Rightarrow \exists$ cvars (not R_c)
$\forall$ vars R $\Rightarrow \forall$ cvars (not R_c)
$\forall$ vars R $\Rightarrow$ (not R_c)

In the illustration above, there is only one negative event, so R_c equals E_4. To see if R_2 is consistent with R_c the proof must be in the last of the three forms above because R_c contains no variables.

Consider our example:

$\forall$ v_1 v_2 x y (convex v_1 v_2 x) (red y) $\Rightarrow$ (not (sphere A)) v (not (red A)) v (not (rectangle B))

This cannot be proved; R_2 is not consistent with E_4. Therefore it may be possible to prove the negation of the consistency constraint, namely that "R generalizes to R_c," in one of two possible forms:

$\exists$ vars R $\Rightarrow \exists$ cvars R_c
$\exists$ vars R $\Rightarrow R_c$

In general let A and B be implications of the forms:

A: $\exists$ vars (R vars) $\Rightarrow$ (R_c vars)
B: $\forall$ vars (R vars) $\Rightarrow$ (not (R_c vars))

There are four cases to consider:

1. A is not proved, and B is proved,
2. A is proved, and B is not proved,
3. A is provable, but not proved and B is not proved, and
4. A is not provable, and B is not proved.

Kodratoff explains that the last situation is a fundamental inconsistency that cannot be repaired. The first case indicates that R is consistent with R_c and thus no repair is needed. In the second case each binding under which the proof holds is a problem of inconsistency between R and R_c. Let the bindings that result from this set of problems be called P. The recognition rule is repaired under this scheme by a specialization that conjoins the form (not (intersects P *binding*)) where *binding* is the set of variable bindings and P is the set of disallowed bindings. The repair for case 3 involves harvesting leftover clauses in two groups LO and LO_c such that one can prove

$\exists$ vars R & LO $\Rightarrow R_c$ & LO_c

Then R can be redefined as R anded with (NOT LO_c), which fits the second case above.

4 SBL AS EBL EXPLANATION REPAIR

Normally one wants rules generated by an SBL algorithm to be consistent and complete. Given n positive events E_i and m negative events F_i, we can take R_c in Kodratoff's approach to be

R_c: G_1 v G_2 v ... v G_n

where the G_i are generalizations of the negative events, and n depends on the degree of generalization performed by structural mapping: n = m and $G_i = F_i$ for no generalization of negative examples; n = 1 when negative examples are described by one generalized conjunctive formula.

Showing consistency requires proving that R is a specialization of (NOT R_c) as is outlined above. Showing completeness requires proving that each positive event is a specialization of R. Using repair mechanisms, one finds a provable theorem that is the conjunction of all the constraints. On finding such a theorem, one has the maximally specific consistent and complete explanation of the class of positive events. Standard EBL generalization can now be used to generalize this result.

Deep bias for generating consistent rules is normally built into SBL learning algorithms. Here it is a constraint on a proof. When this constraint is applied via formal logic, explanation-based generalization (EBL) can generalize against this constraint while giving the effect of SBL learning. It is the placement of the bias that has shifted, and such hard biases as consistency appear to be applicable as a kind of *operationality.*

That this transmutation of algorithms is possible should not be a surprise. But just because it is possible to twist one into performing like the other (to some degree at least) does not mean that doing so will be rewarding. The EBL approach is an incremental approach that fits the notion of *repair* fairly well. However, as mentioned above, it is also convenient to have a technique for generalizing over positive examples while working *against* known negative examples (rather than having to repair again later). This trade-off is one more reason why repair techniques perform the minimum generalization; using more generalization would open them up to the need for more repair.

5 STRENGTHS

There are two fundamental strengths to the method of learning by explanation repair: (1) it is stimulated to learn by failure, and (2) it is based in crisp, formal logic.

The former strength places this approach among general models of human learning that might also turn out to be good AI models. The latter strength helps bring deep biases towards the surface where they will be more obvious in their impact on results and more open to inference over the biases themselves.

The method for explanation repair has another important feature: It can generate multiple candidate repairs and associated explanations. More and more, in inference systems of all kinds, it is important to be able to consider several interpretations under a range of contexts and heuristics. Perhaps this is a small trend towards a more parallel computation model in which several views of the same situation are considered at more or less the same time.

6 POTENTIAL WEAKNESSES AND POSSIBLE EXTENSIONS

It is not clear that consistency and completeness ought to be absolute targets of machine learning. Likewise, explanation repair might not need to be the paramount goal of a system. Many concepts are fuzzy, with probabilistic behavior, and yet humans do not seem perturbed by this. Explanations that are known to fail on anomalous cases might not warrant repair.

When many learning systems are confronted by real-sized problems, they break under combinatorial demands unless they can make wide use of heuristics. The question of how to use heuristic knowledge, or to do plausible rather than complete deductive inference, is important. The method of explanation repair is a straight deductive mechanism on its outside. On the inside lie the critical processes such as the pruning of candidate binding lists for generating repairs. These processes could make widespread use of heuristics, but acquiring, revising, and applying the heuristics is as yet an unsolved problem. The point is this: If the power of the algorithm lies in the performance of interior parts for which good heuristics are critical, then how important or critical is the outside formality, no matter how elegant? Perhaps a formal approach can lead Kodratoff's "AI-learnists" towards elegant techniques for acquiring, applying, and managing heuristic metaknowledge at just the places where today's approaches show it to be critical. That would certainly be a worthy challenge.

References

Ganascia, J.-G. 1985. "Comment oublier a l'aide de contre-exemples," *Actes du congres AFCET RFIA*, Grenoble, November 1985

Michalski, R.S. 1984. "Inductive Learning as Rule [illegible] Transformation of Symbolic Descriptions: A Theory and Implementation," [illegible] *Program Con-*

struction Techniques, Biermann, A.W., Guiho G., Kodratoff, Y. (eds), Macmillan Publishing, pp. 517–552.

Mitchell, T.M. 1983. "Learning and Problem Solving," *Proc. IJCAI-83*, Karsruhe, pp. 1139–1151.

Vrain, C. 1985. "Contre-examples: Explications deduites do l'etude des predicats," *Actes du congres AFCET RFIA*, Grenoble, November 1985, pp. 145–159.

Winston, P.H. 1975. "Learning Structural Descriptions from Examples," *The Psychology of Computer Vision*, Winston, P.H. (ed), Ch. 5, McGraw Hill.