

## **Building Knowledge Bases through Multistrategy Learning and Knowledge Acquisition**

GHEORGHE TECUCI

*Department of Computer Science, George Mason University, Fairfax, VA 22030, USA and  
Center for Artificial Intelligence, Romanian Academy, Bucharest, Romania*

This paper presents a new approach to the process of building a knowledge-based system which relies on a tutoring paradigm rather than traditional knowledge engineering. In this approach, an expert teaches the knowledge based system in much the same way the expert would teach a human student, by providing specific examples of problems and solutions, explanations of these solutions, or supervise the system as it solves new problems. During such interactions, the system extends and corrects its knowledge base until the expert is satisfied with its performance. Three main features characterize this approach. First, it is based on a multistrategy learning method that dynamically integrates the elementary inferences that are employed by the single-strategy learning methods. Second, much of the knowledge needed by the system is generated by the system itself. Therefore, most of the time, the expert will need only to confirm or reject system-generated hypotheses. Third, the knowledge base development process is efficient due to the ability of the multistrategy learner to reuse its reasoning process, as well as the employment of plausible version spaces for controlling the knowledge base development process. This paper illustrates a cooperation between a learning system and a human expert in which the learner performs most of the tasks and the expert helps it in solving the problems that are intrinsically difficult for a learner and relatively easy for an expert.

### **1. Introduction**

Automating the process of building knowledge bases has long been the goal of both Knowledge Acquisition and Machine Learning. The focus of Knowledge Acquisition has been to improve and partially automate the acquisition of knowledge from an expert by a knowledge engineer (Gaines and Boose, 1988; Buchanan and Wilkins, 1992). Knowledge Acquisition has had limited success, mostly because of the communication problems between the expert and the knowledge engineer, which require many iterations before converging to an acceptable knowledge base (KB). In contrast, Machine Learning has focused on mostly autonomous algorithms for acquiring

and improving the organization of knowledge (Shavlik and Dietterich, 1990). However, because of the complexity of this problem, the application of Machine Learning tends to be limited to simple domains. While knowledge acquisition research has generally avoided using machine learning techniques, relying on the knowledge engineer, machine learning research has generally avoided involving a human expert in the learning loop. We think that neither approach is sufficient, and that the automation of knowledge acquisition should be based on a direct interaction between a human expert and a learning system (Tecuci, 1992).

A human expert and a learning system have complementary strengths. Problems that are extremely difficult for one may be easy for the other. For instance, automated learning systems have traditionally had difficulty assigning credit or blame to individual decisions that lead to overall results, but this process is generally easy for a human expert. Also, the “new terms” problem in the field of Machine Learning (i.e. extending the representation language with new terms when these terms cannot represent the concept to be learned), is very difficult for an autonomous learner, but could be quite easy for a human expert (Tecuci and Hieb, 1994). On the other hand, there are many problems that are much more difficult for a human expert than for a learning system as, for instance, the generation of general concepts or rules that account for specific examples, and the updating of the KB to consistently integrate new knowledge.

Over the last several years we have developed the learning apprentice systems Disciple (Tecuci, 1988; Tecuci and Kodratoff, 1990) NeoDisciple, (Tecuci, 1992; Tecuci and Hieb, 1994), Captain (Tecuci et al., 1994) and Disciple-Ops (Tecuci et al., 1995) which use multistrategy learning, active experimentation and consistency-driven knowledge elicitation, to acquire knowledge from a human expert. A significant feature of these systems is their ability to acquire complex knowledge from a human expert through a very simple and natural interaction. The expert will give the learning apprentice specific examples of problems and solutions, explanations of these solutions, or supervise the apprentice as it solves new problems. During such interactions, the apprentice learns general rules and concepts, continuously extending and improving its KB, until it becomes an expert system. Moreover, this process produces verified knowledge-based systems, because it is based on an expert interacting with, checking and correcting the way the system solve problems. These systems have been applied to a variety of domains: loudspeaker manufacturing (Tecuci, 1988; Tecuci and Kodratoff, 1990), reactions in inorganic chemistry, high-level robot planning (Tecuci, 1991; Tecuci and Hieb, 1994), question-answering in geography and, more recently, military command agents for distributed interactive simulation environments (Tecuci et al., 1994), and operator agents for monitoring and repair (Tecuci et al., 1995).

We have also developed a novel approach to machine learning, called *multistrategy task-adaptive learning by justification trees* (MTL-JT), which has a great potential for automating the process of building knowledge-based systems (Tecuci, 1993; 1994). MTL-JT is a type of learning which integrates dynamically different learning strategies (such as explanation-based learning, analogical learning, abductive learning, empirical inductive learning, etc.), depending upon the characteristics of the learning task to be performed. The basic idea of this approach is to regard learning as an inferential process for deriving new or better knowledge from prior knowledge and new input information. Taking this view, one can dynamically integrate the elementary inferences (such as deduction, analogy, abduction, generalization, specialization, abstraction, concretion, etc.) which are the building blocks of the single-strategy learning methods. This MTL method takes advantage of the complementary nature of different single-strategy learning methods and has more powerful learning capabilities.

Based on these results, we have started the development of a new methodology and tool, called Disciple-MTL, for building verified knowledge-based systems which also have the capability of continuously improving themselves during their normal use. A main goal of this research is to provide an expert with a powerful methodology and tool for building expert systems, significantly reducing, or even eliminating the need of assistance from a knowledge engineer. The expert will develop the system by teaching it how to solve problems, in the same way it would teach a human apprentice. This paper presents our current results in developing this methodology, by using an example of building a knowledge-based system for computer workstation allocation and configuration.

This paper is organized as follows. Section 2 presents the general Disciple-MTL methodology for building knowledge-based systems. Section 3 describes the application domain used to illustrate the methodology. Sections 4, 5 and 6 describe and illustrate three main phases of teaching the KB system how to solve problems. Finally, section 7 summarizes the main results obtained and presents some of the future research directions.

## **2. General presentation of the Disciple-MTL methodology**

The process of building a KB system consists of two stages:

- building an initial KB for a general inference engine;
- developing (i.e. extending and correcting) the KB until it satisfies required specifications.

In the first stage the expert, possibly assisted by a knowledge engineer, defines an initial KB (Gammack, 1987). There are two main goals of this stage:

- to allow the expert to introduce into the KB whatever knowledge pieces s/he may easily express;
- to provide the KB system with background knowledge which will support it in acquiring new knowledge from the expert.

During this stage the expert generates a list of typical concepts, organizes the concepts, and encodes a set of rules and various correlations between knowledge pieces expressed as determinations or dependencies (Davies and Russell, 1987; Collins and Michalski, 1989; Tecuci and Duff, 1994), which will permit the system to perform various types of plausible reasoning. This initial KB contains all the knowledge pieces which the expert may easily define, and is usually incomplete and partially incorrect.

In the second stage the KB is developed (i.e. extended and corrected), through training sessions with the expert (which no longer needs any support from a knowledge engineer), until it becomes complete and correct enough to meet the required specifications.

Figure 1 shows the Deductive Closure (DC) of the KB during the second stage of KB development. DC represents the set of problems that can be deductively solved by the system. The solutions of some of these problems may be wrong (because the KB is partially incorrect), while other problems may not be solvable at all (because the KB is incomplete). Figure 1 shows also the Plausible Closure (PC) of the KB during KB development. PC represents the set of problems that can be solved by using plausible inferences (Tecuci and Duff, 1994). Plausible inferences are made by using the rules from the KB not only deductively, but also abductively or analogically. They could also be made by using weaker correlations between knowledge pieces (determinations, dependencies, related facts, etc.). Employing plausible reasoning significantly increases the number of problems that could be solved by the system. In the same time, however, many of the solutions proposed by the system will be wrong. The goal of KB development is to extend and correct the KB of the system until it meets the required specifications. The deductive closure of this final KB is called Acceptable Deductive Closure (AC).

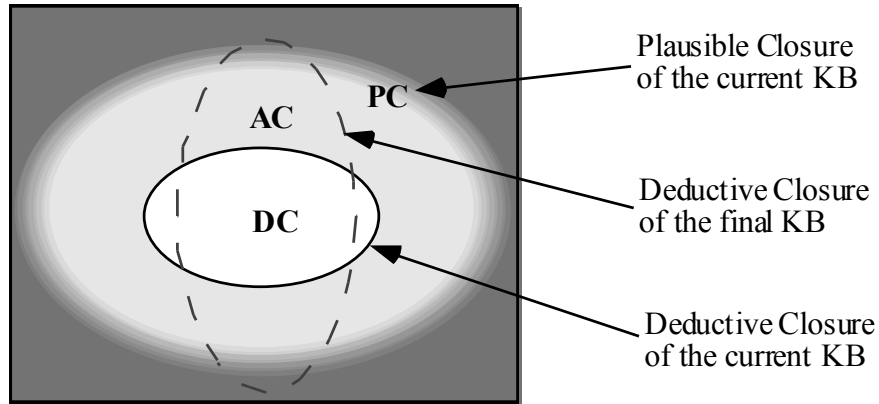


Figure 1: The relationship between DC, PC, and AC during KB development.

As can be seen from Figure 1, the deductive closure during KB development (DC) is an approximate lower bound for AC (the deductive closure of the final KB) in the sense that most of DC is included in AC. Also, the plausible closure during KB development (PC) is an approximate upper bound for AC in the sense that most of AC is included in PC. The set AC is not known to the system. However, any set which includes most of DC and most of which is included in PC is a hypothesis for being the set AC. We can therefore consider PC and DC as defining a *plausible version space* (Tecuci, 1992) which includes the sets that are candidates of being the set AC. With this interpretation, the problem of building the KB reduces to one of searching the set AC in the plausible version space defined by DC and PC. Because the goal is to develop the KB so that its deductive closure becomes AC, DC is used as the current hypothesis for AC, and PC is used as a guide for extending DC, so as to include more of  $PC \cap AC$ , and for correcting DC, to remove from it more of  $DC - AC$ .

The main phases of the KB development process are schematically represented in Figure 2. They are:

- Inference based multistrategy learning,
- Experimentation, verification and repair, and
- Consistency driven knowledge elicitation.

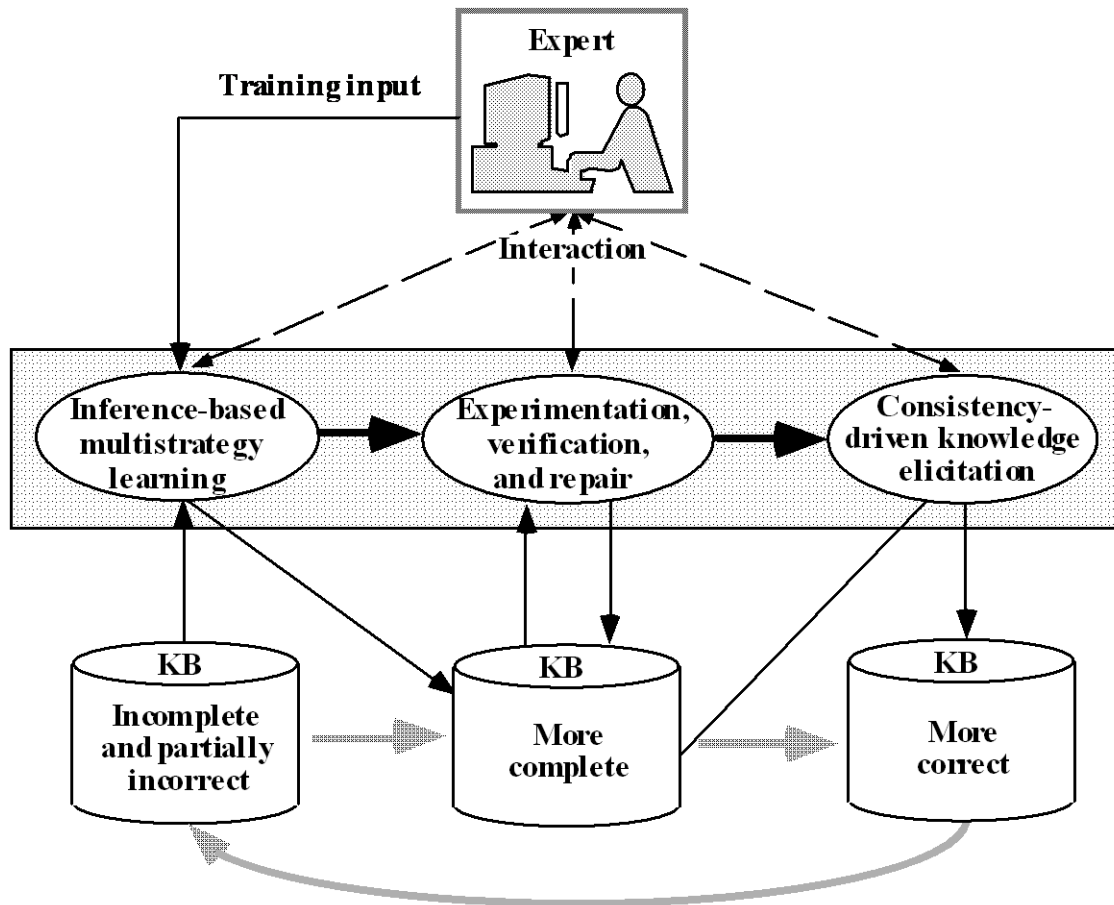


Figure 2: The main phases of the KB development process.

The system is trained by the expert with representative examples of problems and their correct solutions (Training input in Figure 2). Each training problem  $P$  lies generally inside the plausible closure  $PC$  and outside the deductive closure  $DC$  (see Figure 1). This input will initiate a KB development process in which the system will extend and correct its KB so as to correctly solve the problems from the class represented by  $P$ .

In the first phase, *Inference based multistrategy learning*, the system performs a complex reasoning process building “the most plausible and simple” justification tree which shows that the training input indicated by the expert (consisting of a problem  $P$  and its expert-recommended solution  $S$ ) is correct (Tecuci, 1993; 1994). The plausible justification tree is composed of both deductive implications and plausible implications (based on analogy, abduction, inductive prediction, etc.). Because this is the “most plausible and simple” justification tree which show that  $S$  is a correct solution for  $P$ , the component implications and statements are hypothesized to be true, and are asserted into the KB (thus extending  $DC$  with a portion of  $PC$ ).

In the second phase, *Experimentation, verification and repair* (see Figure 2), the system reuses the reasoning process performed in the previous phase in order to further extend, and also to correct the KB so as to be able to solve the problems from the class of which P is an example. The plausible justification tree corresponding to the problem P and its solution S is generalized by employing *various types of generalizations* (not only deductive but also empirical inductive, based on analogy, etc.). Instances of this generalized tree show how problems similar to P may receive solutions similar to S. The system will selectively generate such problems  $P_i$  and corresponding solutions  $S_i$  and will ask the expert to judge if they are correct or not. If the answer is “Yes,” then the KB will be extended, causing DC to include  $P_i$  and its solution. If the answer is “No,” then the wrong implications made by the system will have to be identified, with the expert's help, and the KB will be corrected accordingly. Due to the use of plausible version spaces and a *heuristic search* of these spaces, the system will usually ask a small number of questions.

Because the representation language of the system is incomplete, some of the general problem solving rules learned during experimentation may be inconsistent (i.e. may cover known negative examples). In order to remove such inconsistencies, additional concepts are elicited from the expert, during the third stage called *Consistency driven knowledge elicitation* (see Figure 2).

This KB development process will end when the system has been trained with examples of the main classes of problems it should be able to solve. This should allow the system to solve most of the problems from its area of expertise through deductive reasoning. However, it will also be able to solve an unanticipated problem through plausible reasoning, and to learn from this experience, in the same way it learned from the expert. Therefore, the systems developed using this approach will also have the capability of continuously improving themselves during their normal use.

The KB building process is characterized by a cooperation between the learning system and the human expert in which the learner performs most of the tasks and the expert helps it in solving the problems that are intrinsically difficult for a learner and relatively easy for the expert.

This approach produces *verified knowledge-based systems*, because it is based on verifying and correcting the way the system solves problems. Because this approach is not limited to a certain type of inference engine, one can also use it to build knowledge bases for existing expert system

shells.

The rest of this paper illustrates this methodology.

### **3. Exemplary application domain**

We will use the domain of computer workstation allocation and configuration in order to illustrate our approach to building knowledge bases. The KB system to be built has to reason about which machines are suitable for which tasks and to allocate an appropriate machine for each task. Its inference engine is similar to a Prolog interpreter except that it can perform both deductive and plausible reasoning.

In this example we will use the clausal form of logic. Therefore, instead of writing

“ $A \ \& \ B \ \text{Ø} \ C$ ”, which is read “if A and B are true then C is true”

we will write the equivalent expression

“ $C \text{ :- } A, B.$ ”, which is read “C is true if A and B are true”.

The initial incomplete and partially incorrect KB defined by the expert contains information about various printers and workstations distributed throughout the workplace. A sample of this KB is presented in Figure 3. It contains different types of knowledge: facts, a hierarchy of object concepts, deductive rules, and a plausible determination rule.

The individual facts express properties of the objects from the application domain, or relationships between these objects. For instance, “display(sun01, large)” states that the display of “sun01” is large, and “os(sun01, unix)” states that the operating system of “sun01” is “unix”.

The hierarchy of concepts represents the generalization (or “isa”) relationships between different object concepts. For instance, “microlaser03” is a “microlaser” which, in turn, is a “printer”. These relationships will also be represented by using the notation “isa(microlaser03, microlaser)” and “isa(microlaser, printer)”.

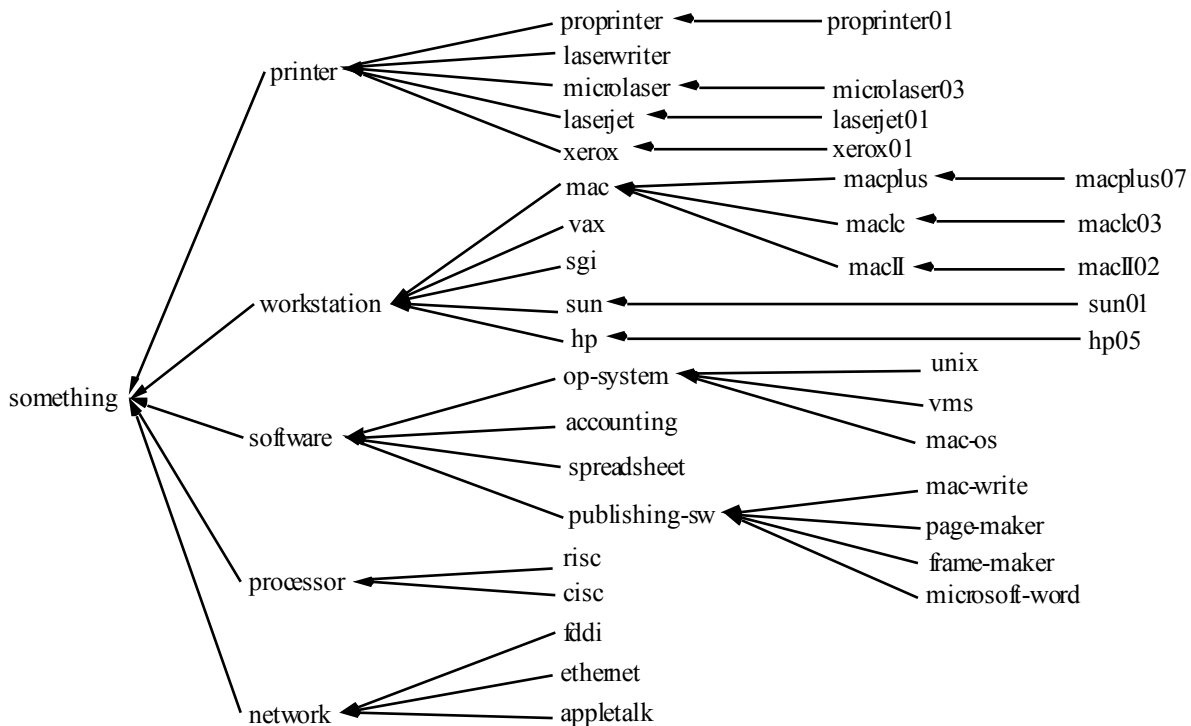
The top of the object hierarchy is “something”, which represents the most general object concept from the application domain. It has the property that “isa(X, something)” is true for any X.



```

os(sun01, unix).    on(sun01, fddi).    speed(sun01, high).  processor(sun01, risc).  display(sun01, large)
; sun01's operating system is unix, it is on the fddi network, has high speed, risc processor and large display
os(hp05, unix).    on(hp05, ethernet).  speed(hp05, high).   processor(hp05, risc).  runs(hp05, frame-maker).
os(macplus07, mac-os).    on(macplus07, appletalk).
os(macII02, mac-os).    on(macII02, appletalk).
os(macIc03, mac-os).    runs(macIc03, page-maker).
on(proprinter01, ethernet).  resolution(proprinter01, high).  processor(proprinter01, risc).
on(laserjet01, fddi).    resolution(laserjet01, high).    processor(laserjet01, risc).
on(microlaser03, ethernet).  resolution(microlaser03, high).  processor(microlaser03, risc).
resolution(xerox01, high).  speed(xerox01, high).            processor(xerox01, risc).
connect(appletalk, ethernet).  connect(appletalk, fddi).        connect(fddi, ethernet).

```



```

suitable(X, publishing) :- runs(X, publishing-sw), communicate(X, Y), isa(Y, high-quality-printer).
; X is suitable for publishing if it runs publishing software and communicates with a high quality printer

communicate(X, Y) :- on(X, Z), on(Y, Z).
; X and Y communicate if they are on the same network

communicate(X, Y) :- on(X, Z), on(Y, V), connect(Z, V).
; X and Y communicate if they are on connected networks

isa(X, high-quality-printer) :- isa(X, printer), speed(X, high), resolution(X, high).
; X is a high quality printer if it has high speed and resolution

runs(X, Y) :- runs(X, Z), isa(Z, Y).

runs(X, Y) :- os(X, Z).
; the type of software which a machine could run is largely determined by its operating system

```

Figure 3: Part of the incomplete and partially incorrect KB for the domain of computer workstation allocation and configuration (the arrows represent IS-A relationships).

An example of a deductive rule is the following one:

$$\text{suitable}(X, \text{publishing}) :- \\ \text{runs}(X, \text{publishing-sw}), \text{communicate}(X, Y), \text{isa}(Y, \text{high-quality-printer}).$$

It states that  $X$  is suitable for desk top publishing if it runs publishing software and communicates with a high quality printer.

The KB in Figure 3 contains also the following plausible determination rule which states that the type of software run by a machine is plausibly determined by its operating system (“:~” means plausible determination):

$$\text{runs}(X, Y) :~ \text{os}(X, Z).$$

After defining an initial KB like the one from Figure 3, the human expert may start training the system by providing typical examples of answers which the system should be able to give by itself. For instance, the expert may tell the system that “macII02” is suitable for publishing:

$$\text{suitable}(\text{macII02}, \text{publishing})$$

This statement is representative of the type of answers which the KB system should be able to provide. This means that the final KB system should be able to give other answers of the form “suitable( $X$ ,  $Y$ )”, where  $X$  is a workstation and  $Y$  is a task.

Starting from this input provided by the expert, the system will develop its KB so that the deductive closure  $DC$  of the KB will include other true statements of the form “suitable( $X$ ,  $Y$ )” and, in the same time, will no longer include false statements of the same form.

The next sections will illustrate this KB development process, following the phases indicated in Figure 2.

## 4. Inference-based multistrategy learning

### 4.1 Input understanding

#### 4.1.1 Building the plausible justification tree

Let us suppose that the expert wants to teach the system which workstations are suitable for desk top publishing. To this purpose, it will give the system an example of such a workstation:

“suitable(macII02, publishing)”

First, the system tries to “understand” (i.e. to explain to itself) the input in terms of its current knowledge by building the plausible justification tree in Figure 4. The root of the tree is the input fact, the leaves are facts from the KB, and the intermediate nodes are intermediate facts generated during the “understanding” process. The branches connected to any given node link this node with facts, the conjunction of which *certainly* or *plausibly implies* the fact at the node, according to the learner's KB. The notion “plausibly implies” means that the target (parent node) can be inferred from the premises (children nodes) by some form of plausible reasoning, using the learner's KB. The branches together with the nodes they link represent individual inference steps which could be the result of different types of reasoning. This tree demonstrates that the input is a plausible consequence of the current knowledge of the system.

The method for building such a tree is a backward chaining uniform–cost search in the space of all AND trees which have a depth of at most  $p$ , where  $p$  is parameter set by the expert. The cost of a partial AND tree is computed as a tuple  $(m, n)$ , where  $m$  represents the number of the deductive implications in the tree, and  $n$  represents the number of the non-deductive implications (which, in the case of the current method, could be obtained by analogy, inductive prediction or abduction). The ordering relationship for the cost function is defined as follows:

$$(m1, n1) < (m2, n2) \text{ if and only if } n1 < n2 \text{ or } (n1 = n2 \text{ and } m1 < m2)$$

This cost function guarantees that the system will find the justification tree with the fewest number of non-deductive implications. In particular, it will find a deductive tree (if one exists) and the deductive tree with the fewest implications (if several exist).

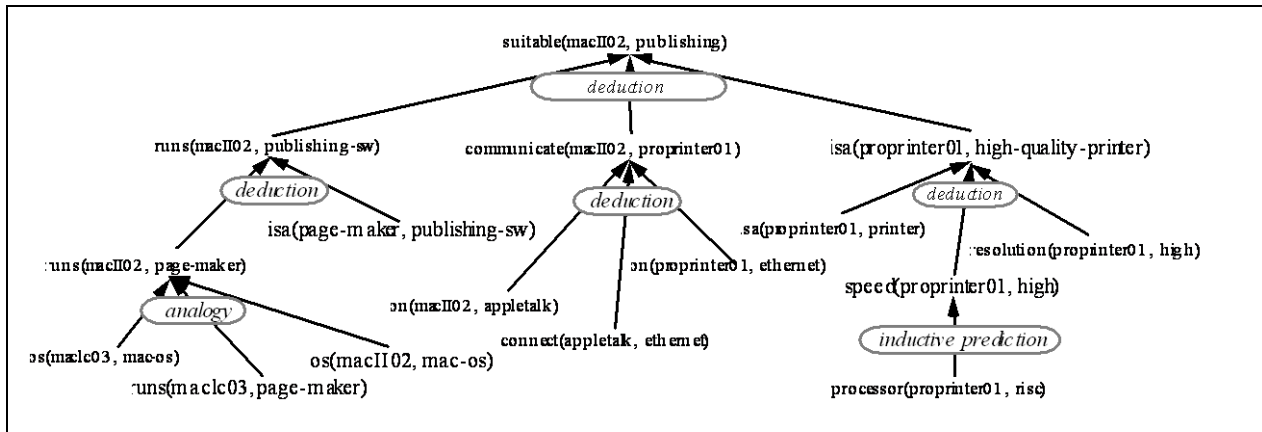


Figure 4: A plausible justification tree for “suitable(macII02, publishing)”.

The tree in Figure 4 is composed of four deductive implications, a determination-based analogical implication and an inductive prediction. It should be noticed that there is no predefined order in the performance of the different inference steps necessary to build a plausible justification tree (Tecuci, 1994). In general, this order depends of the relationship between the KB and the input. Therefore, this method is an example of a dynamic and deep (i.e. at the level of individual inference steps) integration of single strategy learning methods (each learning method corresponding to a specific type of inference).

The next sections present briefly the way the different implications in Figure 4 have been made.

#### 4.1.2 Deduction

Four implications in Figure 4 are the results of deductions based on the deductive rules from the KB in Figure 3, as illustrated in the following.

It is known that

$$\text{suitable}(X, \text{publishing})$$

$$\text{:- runs}(X, \text{publishing-sw}), \text{communicate}(X, Y), \text{isa}(Y, \text{high-quality-printer}).$$

and

$$\text{runs}(\text{macII02}, \text{publishing-sw}).$$

$$\text{communicate}(\text{macII02}, \text{proprinter01}).$$

$$\text{isa}(\text{proprinter01}, \text{high-quality-printer}).$$

Therefore one can conclude that

$$\text{suitable}(\text{macII02}, \text{publishing}).$$

### 4.1.3 Analogy

Analogical inference is the process of transferring knowledge from a known entity S to a similar but less known entity T. S is called the *source* of analogy since it is the entity that serves as a source of knowledge, and T is called the *target* of analogy since it is the entity that receives the knowledge (Winston, 1980; Carbonell, 1986; Gentner, 1990; Gil & Paris, 1995). The central intuition supporting this type of inference is that if two entities, S and T, are similar in some respects, then they could be similar in other respects as well. Therefore, if S has some feature, then one may infer by analogy that T has a similar feature.

A simple analogy is the one based on plausible determinations of the form

$$Q(X, Y) \sim P(X, Z)$$

which is read "Q is plausibly determined by P". This determination states that for any source S and any target T, it is *probably true* that S and T are characterized by a same feature Q (i.e.,  $Q(S, y_0)=\text{true}$  and  $Q(T, y_0)=\text{true}$ ), if they are characterized by the same feature P (i.e.,  $P(S, z_0)=\text{true}$  and  $P(T, z_0)=\text{true}$ ). Therefore, if  $Q(S, y_0)=\text{true}$  then one may infer by analogy that  $Q(T, y_0)=\text{true}$ .

We use the term "probably true" to express that the determination-based analogy we are considering is a weak inference method that does not guarantee the truth of the inferred knowledge. This is different from the determination rules introduced by (Davies and Russell, 1987) which guarantee the truth of the inferred knowledge.

The analogical implication in Figure 4 was made by using the plausible determination rule

$$\text{runs}(X, Y) \sim \text{os}(X, Z)$$

as indicated in Figure 5. According to this rule, the software which a machine can run is largely determined by its operating system. It is known that the operating system of "macIc03" is "mac-os", and that it runs "page-maker". Because the operating system of "macII02" is also "mac-os", one may infer by analogy that "macII02" could also run "page-maker".

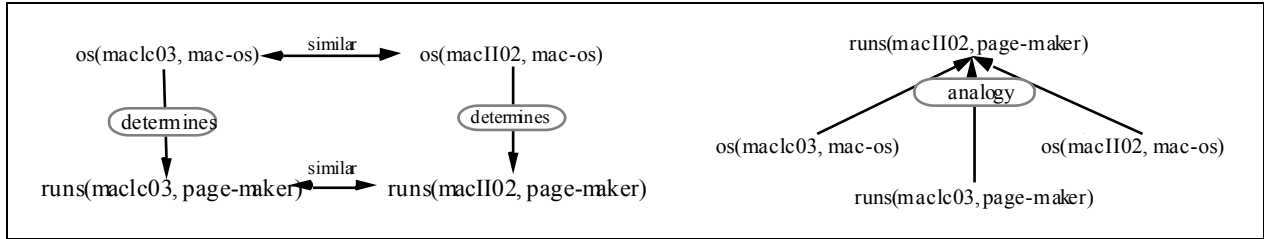


Figure 5: Inferring “runs(macII02, page-maker)” by analogy.

One should notice that a plausible determination rule indicates only what kind of knowledge could be transferred from a source to a target (knowledge about the software which a machine could run, in the case of the considered determination), and in what conditions (the same operating system). It does not indicate, however, the exact relationship between the operating system (for instance, “mac-os”) and the software (“page-maker”). The exact relationship is indicated by the source entity (“macIc03”). Therefore, a plausible determination rule alone (without a source entity), cannot be used in an inference process.

In general, our method of building plausible justification trees is intended to incorporate different forms of analogy, based on different kinds of similarities, such as similarities among causes, relations, and meta-relations (Tecuci, 1994).

#### 4.1.4 Inductive prediction

Inductive prediction is the operation of finding an inductive generalization of a set of related facts and in applying this generalization to predict if a new fact is true.

To build the justification tree in Figure 4, the system needed to prove “speed(proprinter01, high)”. Since there is no implication or determination rule which could be used to make this proof, the system has to analyze the facts from the KB in order to discover a rule which could predict if “speed(proprinter01, high)” is true. The main idea of this process is to look in the KB for other objects X which have high speed, and to take their descriptions as examples of the concept “speed(X, high)”. Then, by generalizing these examples, one learns a rule which could be used to predict if “speed(proprinter01, high)” is true. This process is partially illustrated in Figure 6.

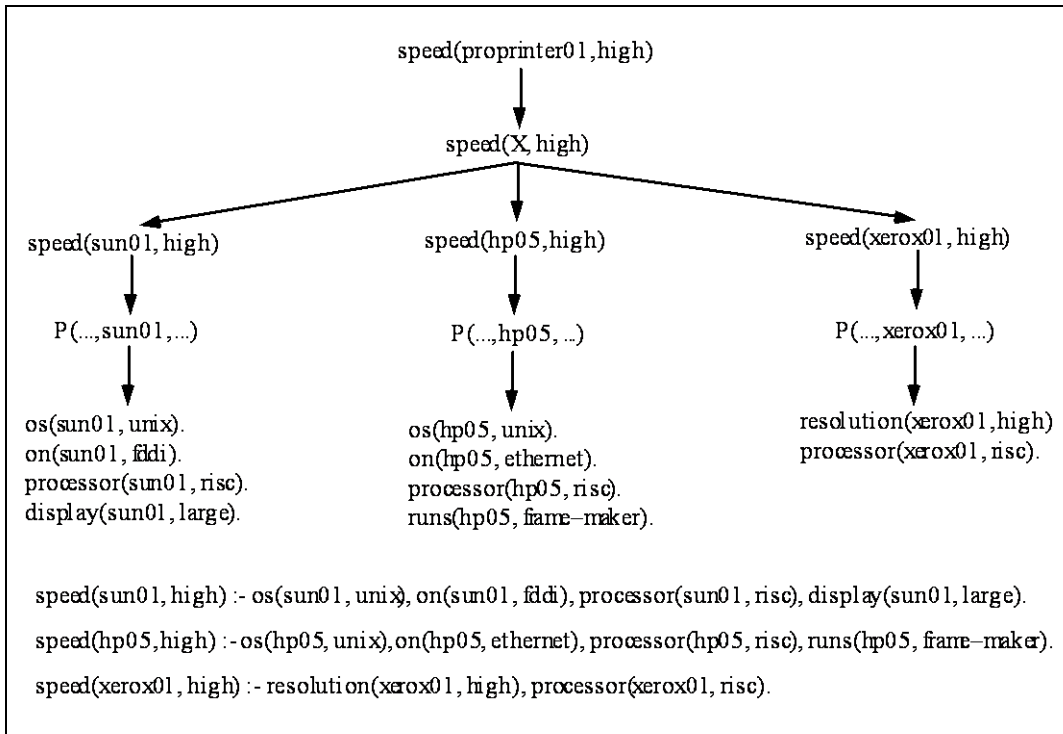


Figure 6: Hypothesizing examples of the concept “speed(X, high)”.

First, the fact “speed(proprinter01, high)” is generalized to “speed(X, high)”. Then, instances of “speed(X, high)” are searched in the KB (see Figure 3). This identifies the objects with high speed. Next, the properties of these objects are determined. Then, the conjunction of the properties of each of these objects is hypothesized as accounting for the high speed of the corresponding object, and as representing an example of the “speed(X, high)” concept (see bottom of Figure 6). The hypothesized concept examples are inductively generalized to the rule

`speed(X, high) :- processor(X, risc).` ; the speed of x is high if its processor is risc

which is applied deductively to infer “speed(proprinter01, high)”.

The above rule is learned by using a decision tree based learning algorithm (Quinlan, 1986). It should also be noticed that the search for the properties of a given object in the KB is guided by schemas characterizing the general form of the rule to be learned, as described in (Lee and Tecuci, 1993).

## 4.2 Improvement of the KB

While there may be several justification trees for a given input, the attempt is to find the most simple and the most plausible one (Tecuci, 1993). This tree shows how a true statement  $I$  derives from other true statements from the KB. Based on the Occam's razor (Blumer et al., 1987), and on the general hypothesis used in abduction which states that the best explanation of a true statement is most likely to be true (Josephson, 1991), one could assume that all the inferences from the most simple and plausible justification tree are correct. With this assumption, the KB is extended by:

- learning a new rule by empirical inductive generalization:  
speed(X, high) :- processor(X, risc).  
*with the positive examples*  
X=sun01. X=hp05. X=xerox01. X=proprinter01.
- learning a new fact by analogy:  
runs(macII02, page-maker).
- discovering positive examples of the determination rule, which is therefore reinforced:  
runs(X, Y) :- os(X, Z).  
*with the positive examples*  
X=macIc03, Y=page-maker, Z=mac-os.  
X=macII02, Y=page-maker, Z=mac-os.
- discovering positive examples of the four deductive rules used in building the plausible justification tree as, for instance:  
suitable(X, publishing) :-  
runs(X, publishing-sw), communicate(X, Y), isa(Y, high-quality-printer).  
*with the positive example*  
X=macII02, Y=proprinter01.

Therefore, the expert merely indicating a true statement allows the system to extend the KB by making several justified hypotheses. As a result of these extensions, the KB entails deductively “suitable(macII02, publishing)”, while before this statement was only plausibly entailed.

During KB development, the rules are constantly updated so as to remain consistent with the accumulated examples. This is a type of incremental learning with full memory of past examples.



### 4.3 Reusing the reasoning process

As mentioned before, the input statement “suitable (macII02, publishing)” is representative for the kind of answers the final KB system should be able to generate. This means that the final KB system should be able to give other answers of the form “suitable(x, y)”. It is therefore desirable to extend *DC* so as to include other such true statements, but also to correct *DC* so as no longer to include false statements of the same form.

Our method is based on the following general explanation-based approach to speed-up learning (Mitchell et al., 1986; DeJong and Mooney, 1986). The system performs a complex reasoning process to solve some problem *P*. Then it determines a justified generalization of the reasoning process so as to speed up the process of solving similar problems  $P_i$ . When the system encounters such a similar problem, it will be able to find a solution just by instantiating this generalization.

The problem solved was the extension of the KB so as to deductively entail the statement “suitable(macII02, publishing)”. This was achieved through a complex multitype inference process of building the plausible justification tree in Figure 4. This reasoning process is generalized by employing various types of generalization procedures (see section 4.4), and then (during the experimentation phase) it is instantiated to various plausible justification trees which show how statements similar to “suitable (macII02, publishing)” are entailed by the KB. Each such plausible justification tree is then used to extend or correct the KB.

## 4.4 Multitype generalization

The plausible justification tree in Figure 4 is generalized by generalizing each implication and by globally unifying all these generalizations. The generalization of an implication depends of the type of inference made, as shown in (Tecuci, 1994). The system employs different types of generalizations (e.g. deductive generalizations, empirical inductive generalizations, generalizations based on different types of analogies, and possibly, even generalizations based on abduction), as will be illustrated in the following sections.

### 4.4.1 Deductive generalization

A deductive implication is generalized to the deductive rule that generated it. This is a *deductive generalization*. For instance

```
suitable(macII02, publishing) :-  
    runs(macII02, publishing-sw),  
    communicate(macII02, proprinter01),  
    isa(proprinter01, high-quality-printer).
```

is generalized to

```
suitable(X1, publishing) :-  
    runs(X1, publishing-sw),  
    communicate(X1, Y1),  
    isa(Y1, high-quality-printer).
```

as shown in Figure 8. The other three deductive implications from the plausible justification tree in Figure 4 are replaced with their corresponding deductive rules, giving new names to the variables from these rules.

### 4.4.2 Generalization based on analogy

An analogical implication is generalized by considering the knowledge used to derive it. To illustrate this type of generalization let us consider again the analogical implication shown in Figure 4 and Figure 5. One could notice that the same kind of reasoning is valid for any machines Z3 and X3, as long as they have the same type of operating system T3, and Z3 runs the software U3. This general analogical reasoning process is illustrated in Figure 7. If one knows

that “os(Z3, T3)”, “os(X3, T3)”, and “runs(Z3, U3)”, then one may infer “runs(X3, U3)”.

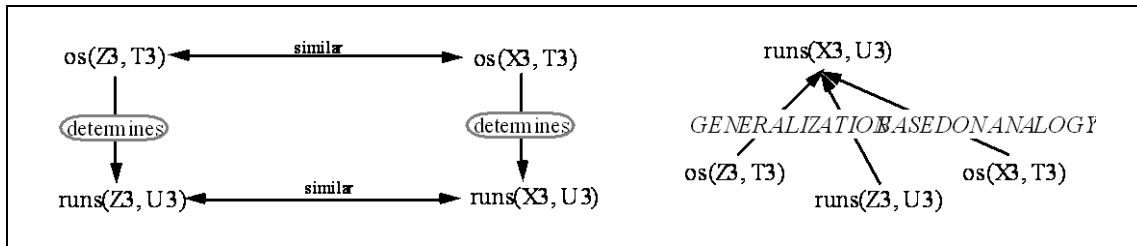


Figure 7: Generalization of the analogical reasoning illustrated in Figure 5.

Consequently, one generalizes the analogical implication from Figure 4 to the general implication from the right side of Figure 7, as shown in Figure 8.

#### 4.4.3 Empirical inductive generalization

An implication obtained through inductive prediction is generalized to the rule that produced it. Therefore, the predicted implication from Figure 4:

speed(proprinter01, high) :- processor(proprinter01, risc).

is generalized to

speed(Y6, high) :- processor(Y6, risc).

as shown in Figure 8.

#### 4.4.4 Generalization of the plausible justification tree

The generalization of the implications from Figure 4 form the explanation structure from Figure 8.

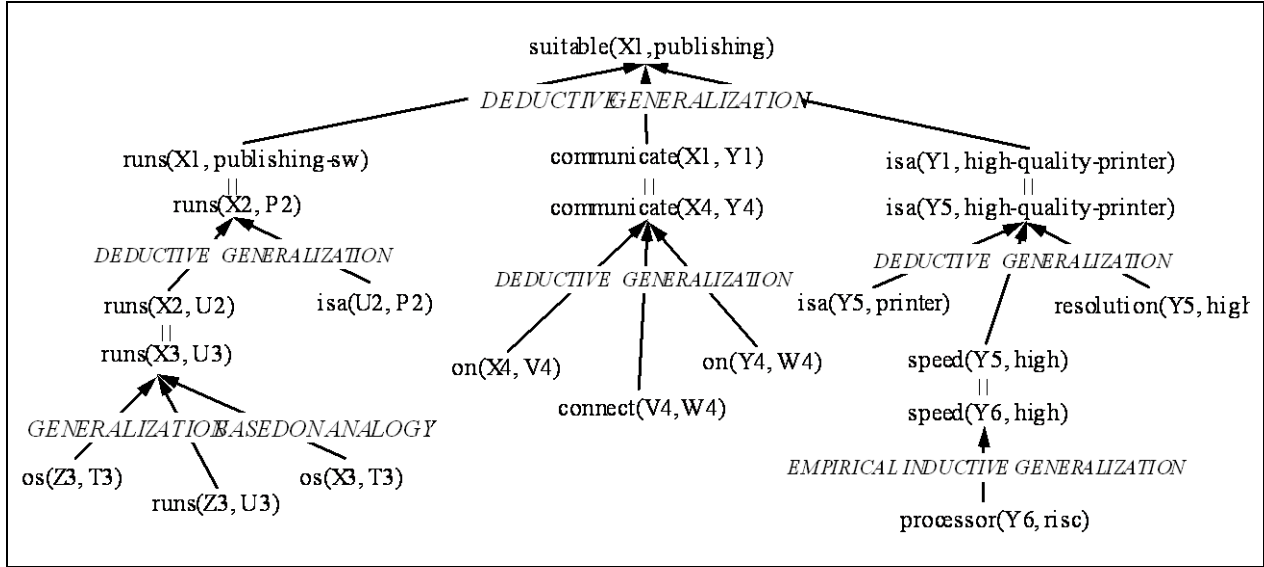


Figure 8: Explanation structure.

To transform this explanation structure into a general justification tree one has to determine the most general unification of the connection patterns, that is, one has to make identical the patterns connected by “||”, as indicated in the following examples:

$$\begin{array}{l}
 \text{runs}(X1, \text{publishing-sw}) \\
 \parallel \\
 \text{runs}(X2, P2) \\
 \\
 \text{runs}(X2, U2) \\
 \parallel \\
 \text{runs}(X3, U3)
 \end{array}
 \qquad
 \begin{array}{l}
 X1 = X2, P2 = \text{publishing-sw} \\
 \\
 X2 = X3, U2 = U3
 \end{array}$$

The most general unification is:

$$(X1 = X2 = X3 = X4, U2 = U3, Y1 = Y4 = Y5 = Y6, P2 = \text{publishing-sw})$$

By applying this unification, and by renaming the variables so that to get rid of the numbers from their names, one obtains the general tree from Figure 9. This tree represents the most general plausible generalization of the justification tree from Figure 4.

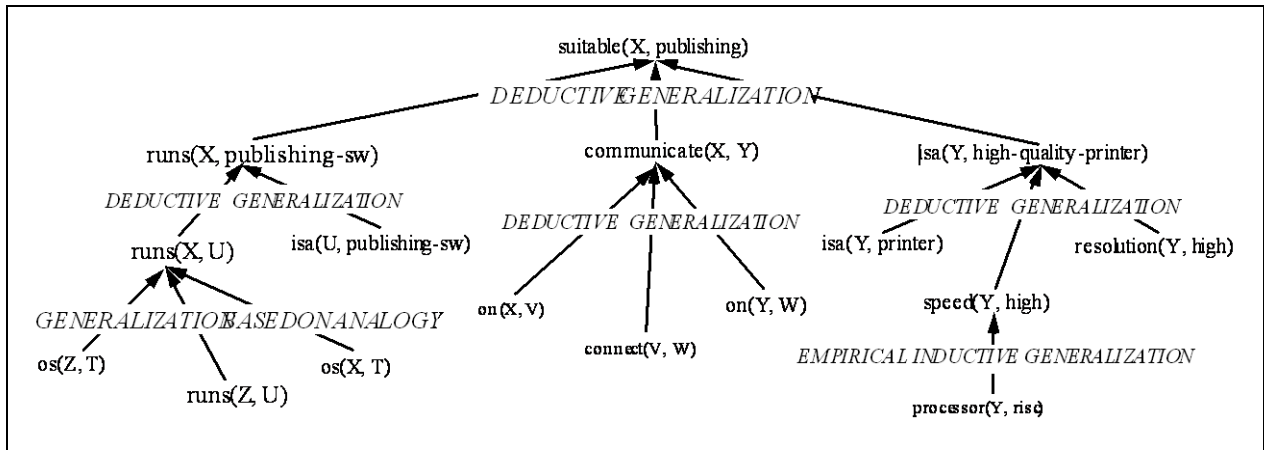


Figure 9: A generalized plausible justification tree.

#### 4.5 An illustration of reusing a past reasoning process

The important feature of the general tree in Figure 9 is that it covers many of the plausible justification trees for statements of the form “suitable(X, publishing)”. If, for instance, “sun01” is another computer for which the leaves of the plausible justification tree in Figure 9 are true, then the system will infer that “sun01” is also suitable for publishing, by simply instantiating the tree in Figure 9. The corresponding instance is the tree in Figure 10.

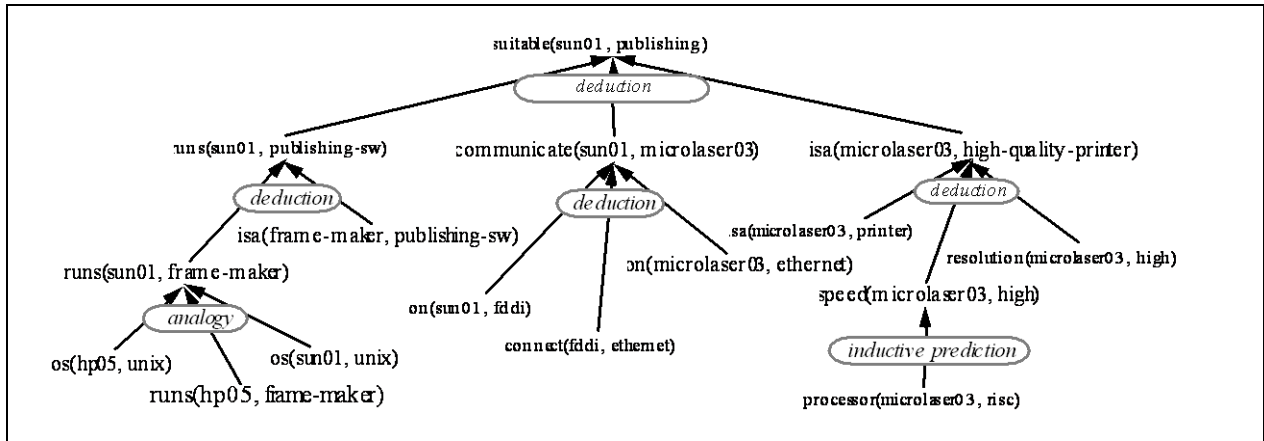


Figure 10: An instance of the plausible justification tree in Figure 9, justifying that sun01 is suitable for publishing.

There are several things to notice when comparing the tree in Figure 4 with the tree in Figure 10:

- although the structure of these trees and the corresponding predicates are identical (both

being instances of the general tree in Figure 9), the arguments of the predicates are different and therefore the meaning of the trees is different;

- while the tree in Figure 4 was generated through a complex and time-consuming reasoning process, the generation of the tree in Figure 10 was a simple matching and instantiation process;
- based on each of these trees the KB is improved in a similar way so as to deductively entail the statement from the top of the tree (assuming that “suitable(sun01, publishing)” is also true).

As will be shown in section 5.2, the expert confirmed that “suitable(sun01, publishing)” is true. Therefore, the KB is improved by:

- discovering a new positive example of the empirically learned rule:

speed(X, high) :- processor(X, risc).

*with the positive examples*

X=sun01. X=hp05. X=xerox01. X=proprinter01. X=microlaser03.

- learning a new fact by analogy:

runs(sun01, frame-maker).

- discovering two new positive examples of the determination rule:

runs(X, Y) :- os(X, Z).

*with the positive examples*

X=maclc03, Y=page-maker, Z=mac-os.

X=macII02, Y=page-maker, Z=mac-os.

X=hp05, Y=frame-maker, Z=unix.

X=sun01, Y=frame-maker, Z=unix.

- discovering a new positive example of each of the four deductive rules used in building the plausible justification tree as, for instance:

suitable(X, publishing) :-

runs(X, publishing-sw), communicate(X, Y), isa(Y, high-quality-printer).

*with the positive example*

X=macII02, Y=proprinter01.

X=sun01, Y=microlaser03.

## 5. Experimentation, verification and repair

Building the plausible justification tree from Figure 4 and its generalization from Figure 9 was the first phase of the KB development process described in Figure 2. The next stage is one of

experimentation, verification, and repair. In this stage, the system will generate plausible justification trees like the one in Figure 10, by matching the leaves of the tree in Figure 9, with the facts from the KB. These trees show how statements of the form “suitable(X, publishing)” plausibly derive from the KB. Each such statement is shown to the expert who is asked if it is true or false. Then, the system (with the expert's help) will update the KB such that it will deductively entail the true statements and only them.

## 5.1 Control of the experimentation

The experimentation phase is controlled by a heuristic search in a plausible version space (*PVS*) which limits significantly the number of experiments needed to improve the KB (Tecuci, 1992). In the case of the considered example, the plausible version space is defined by the trees in Figure 4 and Figure 9, and is represented in Figure 11. The plausible upper bound of the PVS is a rule the left hand side of which is the top of the general tree in Figure 9, and the right hand side of which is the conjunction of the leaves of the same tree. The plausible lower bound of the PVS is a similar rule corresponding to the tree in Figure 4. We call these bounds plausible because they are only approximations of the real bounds (Tecuci, 1992). The plausible upper bound rule is supposed to be more general than the correct rule for inferring “suitable(X, publishing)”, and the plausible lower bound rule is supposed to be less general than this rule.

The plausible version space in Figure 11 synthesizes some of the inferential capabilities of the system with respect to the statements of the form “suitable(X, publishing)”. This version space corresponds to the version space in Figure 1, as indicated in Figure 12. The set of instances of the plausible upper bound corresponds to *PC*, the set of instances of the plausible lower bound corresponds to *DC*, and the set of instances of the correct rule corresponds to *AC*.

*plausible upper bound*

```
suitable(X, publishing) :-
    os(Z, T), runs(Z, U), os(X, T), isa(U, publishing-sw),
    on(X, V), connect(V, W), on(Y, W), isa(Y, printer),
    processor(Y, risc), resolution(Y, high).
```

*plausible lower bound*

```
suitable(macII02, publishing) :-
    os(macIc03, mac-os), runs(macIc03, page-maker), os(macII02, mac-os), isa(page-maker, publishing-sw),
    on(macII02, appletalk), connect(appletalk, ethernet), on(proprinter01, ethernet), isa(proprinter01, printer),
    processor(proprinter01, risc), resolution(proprinter01, high).
```

Figure 11: The plausible version space (*PVS*)

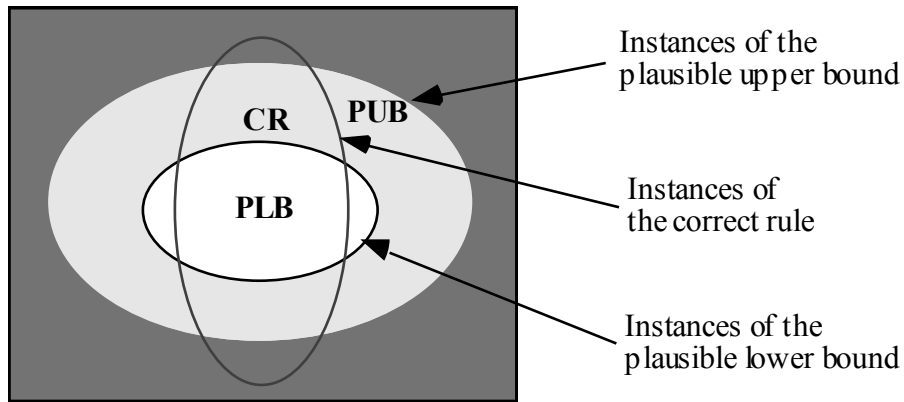


Figure 12: Plausible version space of a rule for inferring “suitable(X, publishing)”.

The plausible version space in Figure 11 is reexpressed in the equivalent, but more operational form from Figure 13. This new representation consists of a single rule the right hand side of which (i.e. the condition) is a plausible version space. The left hand side of the rule in Figure 13 together with the plausible upper bound of the right hand side condition is the same with the plausible upper bound rule in Figure 11 (note that the statements of the form “isa(Q, something)” are always true). The left hand side of the rule in Figure 13 together with the plausible lower bound of the right hand side condition, and the instantiations of the variables of this condition indicated at the bottom of Figure 13, is the same with the plausible lower bound rule in Figure 11.

```

suitable(X, publishing) :-
    plausible upper bound
    isa(T, something), isa(U, publishing-sw), isa(V, something), isa(W, something), isa(X, something),
    isa(Y, printer), isa(Z, something), os(Z, T), runs(Z, U), os(X, T), on(X, V), connect(V, W), on(Y, W),
    processor(Y, risc), resolution(Y, high).

    plausible lower bound
    isa(T, mac-os), isa(U, publishing-sw), isa(V, appletalk), isa(W, ethernet), isa(X, macII02),
    isa(Y, printer), isa(Z, maclc03), os(Z, T), runs(Z, U), os(X, T), on(X, V), connect(V, W), on(Y, W),
    processor(Y, risc), resolution(Y, high).

    with the positive example
    T=mac-os, U=page-maker, V=appletalk, W=ethernet, X=macII02, Y=proprinter01, Z=maclc03.

```

Figure 13: Equivalent form of the plausible version space in Figure 11.

The version space in Figure 13 serves both for generating statements of the form “suitable(X, publishing)”, and for determining the end of the experimentation phase. To generate



such a statement, the system looks into the KB for an instance of the upper bound which is not an instance of the lower bound. Such an instance is the one from Figure 14.

```
suitable(sun01, publishing) :-  
  isa(unix, something), isa(frame-maker, publishing-sw), isa(fddi, something), isa(ethernet, something),  
  isa(sun01, something), isa(microlaser03, printer), isa(hp05, something), os(hp05, unix),  
  runs(hp05, frame-maker), os(sun01, unix), on(sun01, fddi), connect(fddi, ethernet), on(microlaser03, ethernet),  
  processor(microlaser03, risc), resolution(microlaser03, high).
```

Figure 14: An instance of the upper bound of the plausible version space in Figure 13.

Based on this instance, the system generates an instance of the general tree in Figure 9 which shows how “suitable(sun01, publishing)” is plausibly entailed by the KB. This is precisely the tree in Figure 10. The expert is asked if “suitable(sun01, publishing)” is true or false, and the KB is updated accordingly to the expert's answer, as shown in the following sections.

During experimentation, the lower bound of the plausible version space in Figure 13 is generalized so as to cover the generated statements accepted by the expert (the positive examples), and the upper bound (and, possibly, even the lower bound) is specialized so as to no longer cover the generated statements rejected by the expert (the negative examples). This process will end in one of the following situations:

- the bounds of the plausible version space become identical;
- the bounds are not identical, but the KB no longer contains any instance of the upper bound which is not an instance of the lower bound. Therefore, no new statement of the form “suitable(X, publishing)” can be generated.

## 5.2 The case of a true statement that is plausibly entailed by the KB

Because the expert accepted “suitable(sun01, publishing)” as a true statement, the KB and the plausible version space are updated as follows:

- the KB is extended so as to deductively entail “suitable(sun01, publishing)”;
- the plausible lower bound of the PVS is conjunctively generalized to “cover” the leaves of the tree in Figure 10.

The extensions of the KB are similar to those made for the initial expert's input “suitable(macII02, publishing)”, as shown in section 4.5.

The plausible lower bound of the PVS is generalized as shown in Figure 15. For instance, “isa(T, mac-os)” and “isa(T, unix)” are generalized to “isa(T, op-system)”, according to the generalization hierarchy in Figure 3.

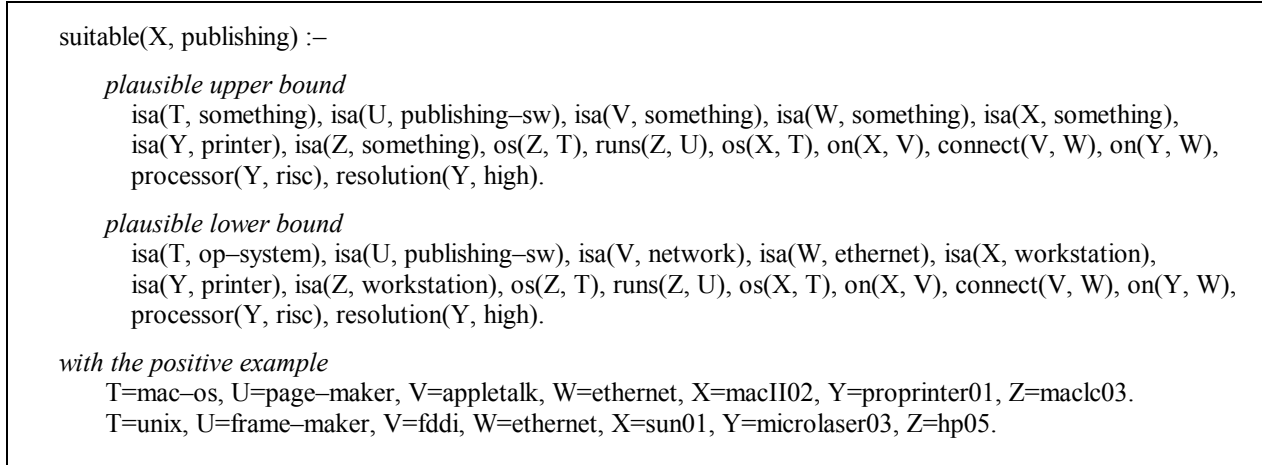


Figure 15: Updated plausible version space (PVS).

### 5.3 The case of a false statement that is plausibly entailed by the KB

Let us also consider the case of a system-generated statement which is rejected by the expert:

“suitable(macplus07, publishing)”.

The corresponding plausible justification tree is shown in Figure 16. This tree was obtained by instantiating the general tree in Figure 9 with facts from the KB. It shows how a false statement is plausibly entailed by the KB. In such a case one has to detect the wrong implication(s) and to correct them, as well as to update the KB, the general justification tree in Figure 9, and the the plausible version space in Figure 15 such that:

- the tree in Figure 16 is no longer a plausible justification tree;
- the KB does not entail deductively “suitable(macplus07, publishing)”;
- the updated general justification tree no longer covers the tree in Figure 16;
- the plausible upper bound of the PVS is specialized so that it no longer covers the leaves of the tree in Figure 16.

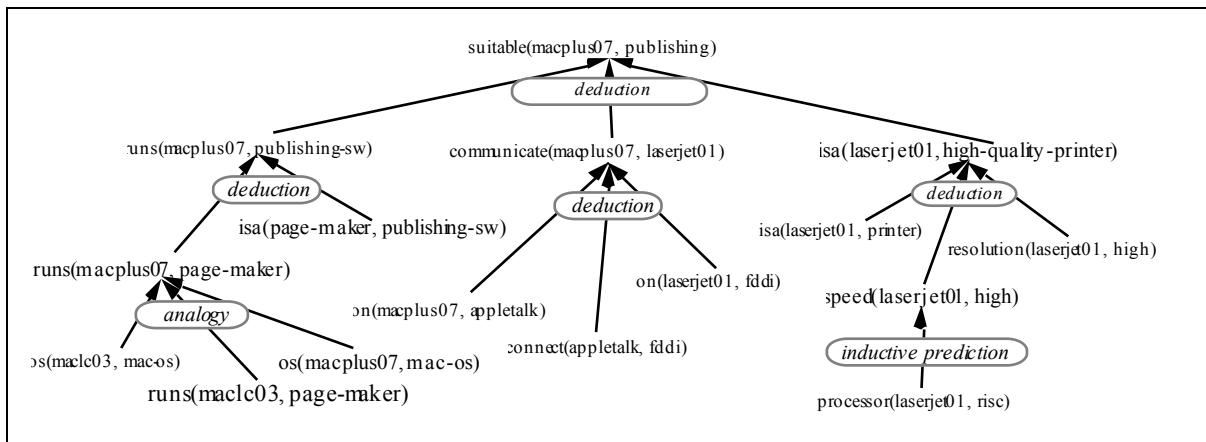


Figure 16: Another instance of the plausible justification tree in Figure 9, which shows how a false statement is plausibly entailed by the KB.

### 5.3.1 Identification of the wrong implication

Detecting the wrong implication from the plausible justification tree in Figure 16 is an intrinsically difficult problem for an autonomous learning system. One possible solution, which is presented in (Tecuci, 1993), is to blame the implication which is the least plausible, and the correction of which requires the smallest change in the KB. For a human expert, however, it should not be too difficult to identify the wrong implication and even to find the explanation of the failure, as shown in (Tecuci, 1992).

In the case of the tree in Figure 16, the wrong implication could be identified by the expert as being the deduction from the top of the tree. Although “macplus07” runs publishing software and communicates with a high quality printer, it is not suitable for publishing because it does not have a large display.

### 5.3.2 Updating the KB, the plausible justification tree and the plausible version space

The rule which generated the wrong implication is specialized as indicated in Figure 17 (requiring X to have a large display). The predicate “display(X, Y)” could be defined by the expert, or could be suggested by the system as being a predicate which could distinguish the known positive examples of the rule from the discovered negative example. This process is described in section 6.

*Wrong implication:*

suitable(macplus07, publishing) :-  
 runs(macplus07, publishing-sw), communicate(macplus07, laserjet01), isa(laserjet01, high-quality-printer).

*Rule which generated the wrong implication:*

suitable(X, publishing) :-  
 runs(X, publishing-sw), communicate(X, Y), isa(Y, high-quality-printer).

*with the positive examples*

X=macII02, Y=proprinter01. X=sun01, Y=microlaser03.

*Updated rule:*

suitable(X, publishing) :-  
 runs(X, publishing-sw), display(X, large), communicate(X, Y), isa(Y, high-quality-printer).

*with the positive examples*

X=macII02, Y=proprinter01. X=sun01, Y=microlaser03.

*with the negative example*

X=macplus07, Y=laserjet01.

Figure 17: Improving a rule based on a negative example.

As a result of rule updating shown in Figure 17, the general plausible justification tree in Figure 9 is updated as shown in Figure 18, and the version space is updated as shown in Figure 19, by adding the predicate “display(X, large)”.

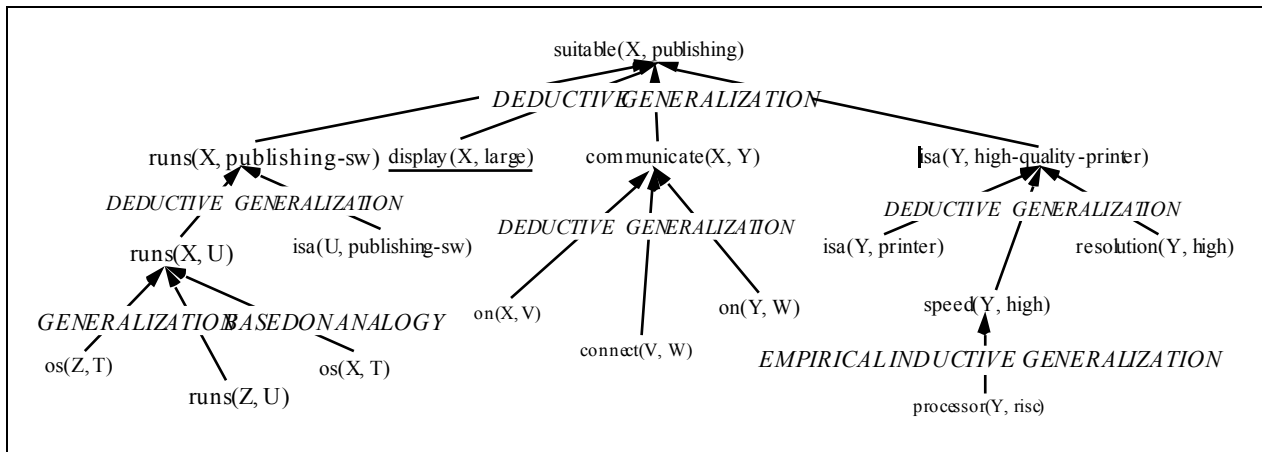


Figure 18: Updated general justification tree.

```

suitable(X, publishing) :-
    plausible upper bound
    isa(T, something), isa(U, publishing-sw), isa(V, something), isa(W, something), isa(X, something),
    isa(Y, printer), isa(Z, something), os(Z, T), runs(Z, U), os(X, T), display(X, large), on(X, V), connect(V, W),
    on(Y, W), processor(Y, risc), resolution(Y, high).

    plausible lower bound
    isa(T, op-system), isa(U, publishing-sw), isa(V, network), isa(W, ethernet), isa(X, workstation),
    isa(Y, printer), isa(Z, workstation), os(Z, T), runs(Z, U), os(X, T), display(X, large), on(X, V), connect(V, W),
    on(Y, W), processor(Y, risc), resolution(Y, high).

with the positive examples
    T=mac-os, U=page-maker, V=appletalk, W=ethernet, X=macII02, Y=proprinter01, Z=maclc03.
    T=unix, U=frame-maker, V=fddi, W=ethernet, X=sun01, Y=microlaser03, Z=hp05.

with the negative example
    T=mac-os, U=page-maker, V=appletalk, W=fddi, X=macplus07, Y=laserjet01, Z=maclc03.

```

Figure 19: Updated plausible version space.

Although the expert may easily identify the wrong implication, it is not as easy for him/her to identify the problem with a wrong implication. In such a case, the wrong implication is kept as a negative exception of the rule which generated it, as shown in Figure 20.

```

suitable(X, publishing) :-
    runs(X, publishing-sw), communicate(X, Y), isa(Y, high-quality-printer).

with the positive examples
    X=macII02, Y=proprinter01.
    X=sun01, Y=microlaser03.

with the negative exception
    X=macplus07, Y=laserjet01.

```

Figure 20: A rule with a negative exception.

## 6. Consistency-driven knowledge elicitation

Because the KB is incomplete and partially incorrect, some of the learned knowledge pieces may be inconsistent (i.e. may cover negative examples), as it is illustrated in Figure 20. In order to remove such inconsistencies, additional knowledge pieces (which may represent new terms in the representation language of the system) are elicited from the expert, through several consistency-driven knowledge elicitation methods, as described in (Tecuci and Hieb, 1994). These methods are applied in the third phase of KB development, as shown in Figure 2. They will be briefly described with the help of the case of the inconsistent rule in Figure 20.

## 6.1 Consistency-driven property elicitation

A negative exception of a rule may be eliminated by identifying (or defining) a new object feature that discriminates between all the covered positive examples and the negative exception.

Let us consider, for instance, rule in Figure 20 that has the covered positive examples represented by

(X = macII02, Y = proprinter01)  
(X = sun01, Y = microlaser03)

and the negative exception represented by

(X = macplus07, Y = laserjet01)

To eliminate the negative exception of the rule, the system will look into the KB for a predicate which could characterize all the positive instances of X (“macII02” and “sun01”), without characterizing the negative exception of X (“macplus07”). If this search is not successful, the system will attempt to elicit such a property from the expert. Alternatively, the system may try to find or elicit a property of the positive instances of Y (“proprinter01”, “microlaser03”), which is not a property of the negative exception of Y (“laserjet01”).

More specifically:

- the system may look in the KB for a property of “macII02” that may be property of “sun01”, (i.e the KB does not deductively imply that “sun01” does not have this property), without being a property of “macplus07”;
- or, it may look for a property of “sun01” that may be property of “macII02”, without being a property of “macplus07”;
- or, it may look for a property of “proprinter01” that may be property of “microlaser03”, without being a property of “laserjet01”;
- or, it may look for a property of “microlaser03” that may be property of “proprinter01”, without being a property of “laserjet01”.

An illustration of this property transfer dialog is shown in Table 1. In this and the subsequent examples the expert’s answers are shown in italics.

Table 1: Sample of a dialog for property transfer.

<p>I know that “display(sun01, large)”.</p> <p>Is it true that “display(macII02, large)” ? [Yes / No / Irrelevant] <i>Yes</i></p> <p>This means that “display(X, large)” is a property characterizing all the covered positive examples and may distinguish them from the negative exception.</p> <p>Is it false that “display(macplus07, large)” ? <i>Yes</i></p>
--

As a result of this dialog, the system has found the property “display(X, large)” which discriminates between the covered positive examples of the rule in Figure 20 and its negative exception. By introducing the discriminating property into the rule’s condition, the negative exception is no longer covered.

Another result of the above dialog is that the descriptions of “macII02” and “macplus07” are refined by adding the elicited property. This is a case of consistency-driven property transfer from one concept (“sun01”) to other concepts (“macII02” and “macplus07”). It is not unusual for an expert to define a property of an object, but then to fail to specify it when describing a related object. With the method presented above, the system may discover and repair such cases of incomplete knowledge.

It may also happen that the system cannot find a property to transfer. In such a case, it will try to elicit a new property by using a technique similar to the triad method employed in the elicitation of the repertory grids (Shaw and Gaines, 1987; Boose and Bradshaw, 1988). If the KB in Figure 3 does not contain the property “display(sun01, large)”, the dialog for eliciting a new property would be as shown in Table 2:

Table 2: Sample of a dialog for property elicitation.

<p>Consider the following groups of objects</p> <p>X: {macII02, sun01} Y: {macplus01}</p> <p>in the context of the following rule</p> <p>suitable(X, publishing) :- runs(X, publishing-sw), communicate(X, Y), isa(Y, high-quality-printer).</p> <p>Could you think of some property that discriminates the objects {macII02, sun01} from the object macplus01 [Yes/No]: <i>Yes</i></p> <p>Describe this property for each object that it applies to:</p> <p><i>display(macII02, large)</i> <i>display(sun01, large)</i> <i>display(macplus01, small)</i></p>
---

## 6.2 Consistency-driven relationship elicitation

A second method for removing the negative exception from the rule in Figure 20 is to look for a relationship between X and Y which could characterize all the positive instances of X and Y, without characterizing the negative exception. That is, the system will look for a relationship between “macII02” and “proprinter01” which could also hold between “sun01” and “microlaser03”, without holding between “macplus07” and “laserjet01”. Or, it could look for a relationship between “sun01” and “microlaser03” which could also hold between “macII02” and “proprinter01”, without holding between “macplus07” and “laserjet01”. Or it could try to elicit such a relationship from the expert.

## 6.3 Consistency-driven concept elicitation

The third method for removing the negative exception from the rule in Figure 20 is similar to the one described in (Wrobel, 1989). It consists of trying to elicit a new concept that covers all the positive instances of X (“macII02” and “sun01”) without covering the negative exception of X (“macplus07”). Alternatively, one could try to elicit a concept that covers all the positive instances of Y (“proprinter01” and “microlaser03”) without covering the negative exception of Y (“laserjet01”).



## 7 Conclusions

We have presented a methodology for building KB systems for a general inference engine. This methodology, which integrates multistrategy learning, active experimentation, and guided knowledge elicitation, emerged from two related research directions:

- the knowledge acquisition methodology of Disciple (Tecuci, 1988; Tecuci and Kodratoff, 1990) and NeoDisciple (Tecuci, 1992; Tecuci and Hieb, 1994);
- the multistrategy task-adaptive learning method based on plausible justification trees MTL-JT (Tecuci, 1993; 1994).

On the one hand, it extends Disciple and NeoDisciple with respect to the knowledge representation used and the types of inference and generalization methods employed and, on the other hand, it adapts and integrates the MTL-JT method into an interactive knowledge acquisition scenario.

One important feature of the presented methodology is the use of plausible justification trees which dynamically integrate the elementary inferences employed by the single-strategy learning methods (such as deduction, analogy, abduction, generalization, specialization, abstraction and concretion), depending of the system's knowledge and the system's input. This allows the system to learn as much as possible from every input received from the human expert, generating by itself much of the knowledge needed to develop the KB. It is very similar in spirit to the knowledge acquisition methodology of (Gil & Paris, 1995) where much of the knowledge needed by the system is derived by analogy with the existing knowledge. This feature of our methodology provides it with a significant advantage over the traditional manual knowledge acquisition methods in which the expert provides all the needed knowledge to the system. It is also important to stress that the plausible justification tree represents itself a general framework for integrating a whole range of inference types (Tecuci, 1993). Therefore, theoretically, there is no limit with respect to the type or number of inferences employed in building such a tree.

Another important feature of the methodology is the employment of different types of generalizations. The current machine learning research distinguishes only between deductive generalizations and inductive generalizations. Our research shows that one could make much finer distinctions by associating a specific generalization with each type of inference as, for instance, a generalization based on analogy. The generalization of an inference process synthesizes the assumptions supporting the inference and may therefore allow reusing that inference process in new situations. This feature is exploited by our methodology, which reuses

the KB development processes, and is therefore very efficient.

The efficiency of the presented methodology is also due to the use of simple plausible version spaces and a human guided heuristic search of these spaces. The concept of plausible version space is both a generalization and an operationalization of the classical version space concept introduced by Mitchell (1978).

Finally, this methodology is based on a cooperation between a learning system and a human expert in which the learner performs most of the tasks and the expert helps it in solving the problems that are intrinsically difficult for a learner and relatively easy for an expert. In particular, the expert solves the blame assignment problem, as illustrated in section 5.3, and the new terms problem, as illustrated in section 6.

There are also several ways in which the presented methodology could be improved. First of all, the set of inferences involved is quite limited (deduction, determination-based analogy, inductive prediction, and abduction). New types of inference should be included, as well as more complex versions of the current ones. Consequently, new types of justified generalizations, corresponding to these new types of inference, should also be defined. Another issue is to determine a good measure for comparing the plausibility of different justification trees, so the system only hypothesizes the most plausible knowledge.

The consistency-driven knowledge elicitation methods briefly presented in section 6 could be extended so as, not only to add new concepts and relationships into the KB, but also to delete those that become unnecessary.

Another future research direction consists in developing a qualitative representation of the certainty of the knowledge pieces from the KB (e.g. some facts are characterized as true by the expert, while others are hypothesized as true by the system, some rules are initially defined by the expert, while others are learned by the system). In particular, one has to be able to estimate the confidence in the learned rules, to update the confidence of a rule when new examples or exceptions are discovered, as well as to maintain only a limited number of “representative” examples of a rule (during KB development) so that neither to overload the system, nor to lose important information.

## Acknowledgements

David Duff contributed to the development of the application domain presented in this chapter.

## References

- Blumer A., Ehrenfeucht A., Haussler D. and Warmuth M. K. (1987). Occam's Razor, *Information Processing Letters*, 24, pp. 377–380.
- Boose J.H. and Bradshaw J.M. (1988). Expertise Transfer and Complex Problems: Using AQUINAS as a knowledge-acquisition workbench for knowledge-based systems, in *Knowledge Acquisition Tools for Expert Systems*, J. Boose and B. Gaines (Eds.), Academic Press.
- Buchanan B.G. and Wilkins D.C., Eds. (1992). Readings in Knowledge Acquisition and Learning: Automating the Construction and Improvement of Expert Systems, Morgan Kaufmann, San Mateo, CA.
- Carbonell J.G. (1986). Derivational Analogy: a Theory of Reconstructive Problem Solving and Expertise Acquisition, in *Machine Learning: An Artificial Intelligence Approach*, Vol. 2, R.S. Michalski, J.G. Carbonell and T.M. Mitchell (Eds), San Mateo, CA: Morgan Kaufmann.
- Collins A. and Michalski R.S. (1989). The Logic of Plausible Reasoning: A Core Theory, *Cognitive Science*, 13, pp. 1–49.
- Davies T.R. and Russell S.J. (1987). A Logical Approach to Reasoning by Analogy, *Proc. IJCAI*, Milan, Italy, Morgan Kaufmann.
- DeJong G. and Mooney R. (1986). Explanation-Based Learning: An Alternative View, *Machine Learning*, 1, pp. 145-176.
- Gaines B. R. and Boose J. H., Eds (1988). *Knowledge Acquisition for Knowledge Based Systems*, Academic Press.
- Gammack J.G. (1987). Different Techniques and Different Aspects on Declarative Knowledge, in Alison L. Kidd, Ed. *Knowledge Acquisition for Expert Systems: A Practical Handbook*, Plenum Press.
- Gil Y. and Paris C. (1995). Towards Method-Independent Knowledge Acquisition, in *Machine Learning and Knowledge Acquisition: Integrated Approaches*, G. Tecuci and Y. Kodratoff (Eds), Academic Press.
- Gentner D. (1990). The Mechanisms of Analogical Reasoning, in *Readings in Machine Learning*, J.W.Shavlik and T.G.Dietterich (Eds), Morgan Kaufmann.
- Josephson J. (1991). Abduction: Conceptual Analysis of a Fundamental Pattern of Inference,

- Research Report 91-JJ*, Laboratory for AI Research, Ohio State University.
- Lee O. and Tecuci G. (1993). MTL: An Inference-based Multistrategy Learning System, in *Proc. InfoScience 93*, Korea, October 21-22.
- Mitchell T.M. (1978). Version Spaces: An Approach to Concept Learning, *Doctoral Dissertation*, Stanford University.
- Mitchell T.M., Keller T. and Kedar-Cabelli S. (1986). Explanation-Based Generalization: A Unifying View, *Machine Learning*, 1, pp. 47-80.
- Quinlan J. R. (1986). Induction of Decision Trees, *Machine Learning* 1, pp. 81-106.
- Shavlik J. and Dietterich T., Eds. (1990). *Readings in Machine Learning*, Morgan Kaufmann, San Mateo, CA.
- Shaw M.L.G. and Gaines B.R. (1987). An interactive knowledge elicitation technique using personal construct technology, in *Knowledge Acquisition for Expert Systems: A Practical Handbook*, A.L. Kidd (Ed.), Plenum Press.
- Tecuci G. (1988). Disciple: A Theory, Methodology, and System for Learning Expert Knowledge, *Ph.D. Thesis*, University of Paris-South.
- Tecuci G. (1991). A Multistrategy Learning Approach to Domain Modeling and Knowledge Acquisition, in Y.Kodratoff (ed), *Machine Learning EWSL-91*, Springer-Verlag.
- Tecuci G. (1992). Automating Knowledge Acquisition as Extending, Updating, and Improving a Knowledge Base, *IEEE Trans. on Systems, Man and Cybernetics*, 22, pp. 1444-1460.
- Tecuci G. (1993). Plausible Justification Trees: a Framework for the Deep and Dynamic Integration of Learning Strategies, *Machine Learning*, 11, pp. 237-261.
- Tecuci G. (1994). An Inference-Based Framework for Multistrategy Learning, in R.S. Michalski and G. Tecuci (Eds), *Machine Learning: A Multistrategy Approach*, vol. 4, Morgan Kaufmann.
- Tecuci G. and Kodratoff Y. (1990). Apprenticeship Learning in Imperfect Theory Domains, in *Machine Learning: An Artificial Intelligence Approach*, Vol. 3, Y. Kodratoff and R.S. Michalski (Eds.), San Mateo, CA, Morgan Kaufmann.
- Tecuci G. and Hieb M. (1994). Consistency-driven Knowledge Elicitation: Using a Learning-oriented Knowledge Representation that Supports Knowledge Elicitation in NeoDisciple, *Knowledge Acquisition Journal*, 6, pp. 23-46.
- Tecuci, G. and Duff, D. (1994). A Framework for Knowledge Base Refinement through Multistrategy Learning and Knowledge Acquisition. *Knowledge Acquisition Journal*, 6, pp. 137-162.
- Tecuci G., Hieb M.H., Hille D. and Pullen J.M. (1994) Building Adaptive Autonomous Agents for Adversarial Domains, in *Proceedings of the AAAI Fall Symposium on Planning and Learning*, November, New Orleans.

- Tecuci G., Hieb M.H. and Dybala T. (1995). Building an Adaptive Agent to Monitor and Repair the Electrical Power System of an Orbital Satellite, in *Proceedings of the Goddard Conference on Space Applications of Artificial Intelligence and Emerging Information Technologies*, NASA Goddard, Greenbelt, Maryland, May.
- Winston P.H. (1980). Learning and Reasoning by Analogy, *Communications of the ACM*, Vol. 23.
- Wrobel S. (1989). Demand-driven Concept Formation, in Morik K. (Ed.), *Knowledge Representation and Organization in Machine Learning*, Springer Verlag.