(KROML) and for making many constructive suggestions about the paper.

References:

Adelson, B. & Soloway, E. A model of software design. Technical Report #342, Department of Computer Science, 1984.

Letovsky, S. Cognitive processes in program comprehension. In E. Soloway & S. Iyengar (Eds.) Empirical studies of programmers, Ablex, Norwood, NJ, 1986.

Littman, D., Pinto, J., Letovsky, S., & Soloway, E. Mental models and software maintenance. In E. Soloway & S. Iyengar (Eds.) Empirical studies of programmers, Ablex, Norwood, NJ, 1986.

Littman, D. & Soloway, E. Mental simulation of programs: Some Data and some theory. Unpublished paper (1986) available from the author.

Littman, D. Modelling human expertise in knowledge engineering: Some preliminary observations. IJMMS 1987.

Sloppy Modeling

Katharina Morik
Technical University Berlin
Project Group KIT
Sekr. FR 5-8
Franklinstr.28/29
D-1000 BERLIN 10
<MORIK@DB0TUI11.BITNET>

Abstract

In this paper, I would like to present a unifying view on knowledge acquisition and machine learning. In this view, knowledge acquisition systems should support the user in doing the modeling of a domain, and machine learning systems are those which perform part of the modeling autonomously. Taking the notion of modeling as the central point, some aspects of modeling along with their impact for building knowledge acquisition and machine learning systems are discussed. In particular, reversability at all levels is claimed to be supported by the system.

As a result of the unifying view, a new way of integrating machine learning into knowledge acquisition is presented and exemplified by the system BLIP [1], a system which supports the user in domain modeling and at the same time takes part of the work off the user's back by modeling autonomously. Since all decisions regarding the model can be revised and revision is supported by the system, we call this way of modeling "sloppy modeling"; the user may start with a sloppy model which can be revised and enhanced.

1 Knowledge Acquisition as Transfer

Knowledge acquisition and knowledge engineering has been viewed as a transfer process for a long time (Hayes-Roth, Waterman, Lenat 83). This view is best illustrated by the bottleneck metaphor cited all over the literature: knowledge acquisition is the bottleneck of building expert systems. Knowledge engineering then means extracting knowledge out of the expert and pushing a completed model into the expert system. We can assign two interpretations to this metaphor: on the one hand, the expert system is the bottle, and the problem is to get the knowledge in. On the other hand, the expert is meant by the bottle, and the problem is to get the knowledge out.

---

[1] BLIP is currently under development at the Technical University Berlin, project KIT-Lerner; KIT-Lerner is partially supported by the German ministry for research and technology (BMFT) under contract ITW8501B1. Industrial partners are Nixdorf Computer AG and Stollmann GmbH.

1.1 The Expert System as a "Bottle"

First-generation expert systems (like MYCIN) offer the user only one data structure to represent knowledge about a domain, knowledge about a task, and knowledge about consultation strategy: the production rule. Knowledge engineering within this framework means encoding diverse types of knowledge as production rules. In order to support the knowledge engineer, tools such as knowledge editors, debuggers, and menu interfaces have been developed.

These tools are supposed to serve as "funnels." They are built to offer an additional functionality to the expert system shell. In particular, they should provide for

- inspectability of the knowledge base,
- explainability of system behavior, and
- changeability of the encoded knowledge.

However, as will be shown in the next section of this paper, tools cannot help the knowledge engineer more than the expert system's representation language allows them to.

1.1.1 Tools don't help

In this section, the shortcomings of knowledge acquisition with respect to first-generation expert systems are discussed using three examples: goal integration, representation at the most specific level (object level), and uniform representation of diverse knowledge types. The examples have already been discussed with respect to explainability of expert systems (Swartout 83; Neches, Swartout, Moore 85; Clancey 86). Here, I want to recall this discussion and point out its impact on knowledge engineering.

The problem of goal integration or integration of action parts of production rules has been raised by Mostow and Swartout (86). Usually, different derivations of the same conclusion are integrated by strengthening the certainty factor of the conclusion using Shortliffe's (Shortliffe 76) formula

$$1-\prod_{i=1}^{n}(1-x_i),$$

where xi are the certainty factors of the single conclusions.
This formula is only applicable if the premises of the rules are mutually independent. Swartout takes this example and proposes an explicit representation of goals and methods. I do not want to go that far here, but only indicate consequences for tools. Adding a new rule thus requires checking all rules with the same action part, looking for specialization relations between premises, and preventing these relations. Since the system has no epistemic knowledge of super- and subconcepts, it is up to the knowledge engineer to take care that the premise of the new rule is independent of already existing premises with the same conclusion. A knowledge acquisition tool can display all rules with the same action part. It cannot, however, watch over the independence of premises. This example already shows that a tool cannot offer a really additional functionality to the expert system shell. In fact, a good tool requires functionality on the part of the expert system shell itself; here: epistemic knowledge of super- and subconcepts.

The representation at the most specific level in EMYCIN-alike systems has been criticized by, among others, Clancey (86). The prominent example is the representation of heuristics in MYCIN. MYCIN behaves as if there were the following heuristic:

If the genus of a micro-organism can be determined but not its species
then take the most probable species for that genus.

This behavior, however, is a result of rules which give the most probable species for a genus. The heuristic is encoded as the level of "genus" and "species". Therefore, the heuristic cannot be found and changed at one place in the knowledge base. Moreover, the system cannot collect all rules involved in the heuristic in order to display them to the user as it could - in the first example - show all rules with the same action part. Neither can the user express her/his wish to change the heuristic nor can the system determine the set of rules corresponding to the heuristic. Another consequence of this representational framework is that changing a rule on species may cause a new system behavior in a way the user is not aware of. No tool can support the user in changing the heuristic or call the user's attention to consequences of changing innocent-looking rules for the system's overall behavior.

The uniform representation of consultative, problem-solving, and domain knowledge prevents the user from changing just one type of knowledge. For example, the strategy of questioning the advice-seeking user cannot be changed as such. Moreover, changing a rule system behavior in an unforeseen way. The well-known rule of MYCIN:

If the age of the patient is greater than 17 and
the patient is an alcoholic
then Diplococcus might be causing infection.

is of the same form as

If the age of the patient is less than 7
then remove Tetracycline from the list of drugs under consideration.

(Clancey 86). The intention behind introducing the age of the patient into the left-hand side of the rule, however, is different. In the first rule, younger patients are not asked whether they are alcoholics because of the first premise. In the second rule, Tetracycline are not prescribed to children, because Tetracycline can do harm to their teeth and bones. If child alcohol abuse increases so that the question of whether the patient is an alcoholic is also applicable to the 12-year-olds, a rule on Diplococcus but not a similar looking rule on Tetracycline has to be changed. How can the knowledge engineer find the rules to be changed? How could a tool find them for him?
In addition, the knowledge engineer inspecting knowledge about Diploccocus may well delete the age-premise of the first rule, regarding it as nonsense. He then changes the questioning behavior, without intending it and neither the system nor a tool can warn him.

As these examples show, even tools with high polished interfaces using window and mouse techniques cannot compensate for representational deficiencies of the expert system shell. In terms of the metaphor: building funnels is no solution if the bottle is the problem.

1.1.2 Newer Acquisition Systems

The discussion sketched in the previous section has led to newer developments in expert systems. As tools turned out to be inappropriate, work concentrated on developing expert system shells which ease knowledge acquisition by their representation forms. The goal is to offer representational constructs which match the expert's concepts. It is the standard AI goal of representing knowledge in the "human window" (Michie 82). Where production rules were first viewed as corresponding to expert's concepts they now are classified as "implementation-level primitives" to a domain ontology (e.g. "disease process taxonomy") and epistemological distinctions. The underlying assumption with respect to knowledge acquisition is: if there is no mismatch between representational constructs offered by the system and mental concepts to the expert, knowledge acquisition is more or less a direct typing-in process.

A system which allows the user to type her/his knowledge into the system in the same way as s/he used to write it on paper is OPAL (Musen, Fagan, Combs, Shortliffe 87). The system acquires oncological reports like with spread-sheets. This is possible because the oncological knowledge is already fully represented. The domain model is used to guide the acquisition dialog. Turning it the other way around: the hard problem of domain modeling must have been solved before the easy knowledge acquisition such as filling in forms becomes applicable. Similarly, the KRIMB system (Cox, Blumenthal 87) reduces knowledge acquistion to the specification of an already prepared model. For instance, it is already known that animals eat food, that cats are animals, that fish is food, and that cats eat fish. The system is then capable of acquiring the fact that a particular cat eats a particular fish. Calling the specification process "model building" is misleading.

A more general knowledge acquisition tool is MOLE (Eshelman, Ehret, McDermott, Tan 87). MOLE also exploits assumptions about the world. These assumptions, however, are much more general than those of OPAL or KRIMB. It is presupposed that

- the task of the new domain is of the heuristic classification type,
- there are hypotheses which cover symptoms,
- there are symptoms which differentiate between hypotheses, and
- there is knowledge about the combination of hypotheses.

This general structure is adequate for a wide range of domains. It enables the system, e.g. to check whether hypotheses can be differentiated in principle, to supply default support values, to discover intermediary concepts between symptoms and hypotheses. In a second phase of knowledge acquisition, the knowledge base is interactively refined. In this framework, the knowledge engineer is regarded an AI programmer. He is offered a workbench with which he can encode a knowledge base. This is analogous to the environment of a programming language. As is the case there, the model (algorithm) has to be constructed by the knowledge engineer (programmer); only the encoding is supported. In fact, model building is not at all supported by these systems but rather presupposed.
Figure 1 shows the transfer view of knowledge acquisition.

The newer approaches differ from the tool approach in the integration of knowledge acquisition and the performance element. The two approaches do not differ, however, in the sequential procedure of

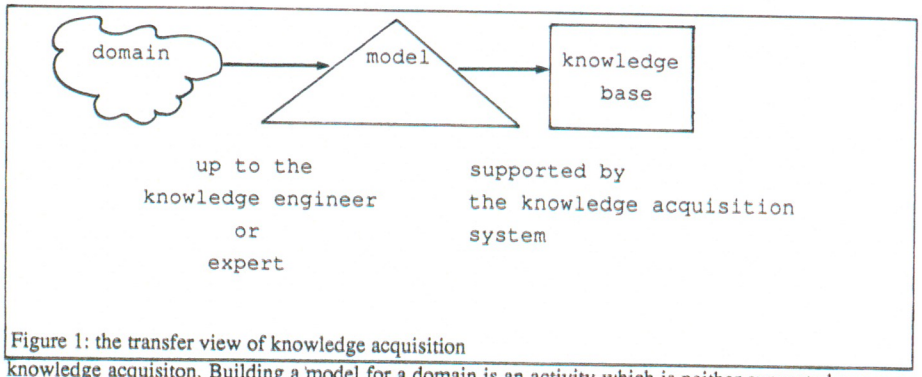


Figure 1: the transfer view of knowledge acquisition

knowledge acquisiton. Building a model for a domain is an activity which is neither supported nor even recognized by both approaches. The model is viewed as given in a more or less completed form. The problem is then to transfer the given model from one representation form (on paper, in the head) into another (the representation form of the system). The influence of the representations and the influence of the encoding phase back to the model building cannot be dealt with in this paradigm. In terms of figure 1, the arrow back from the knowledge base to the model and model building is missing.

In particular, revision of the model based on experience with the system's performance is restricted to refinement. Only belief revision in the knowledge base is supported by the system, but changing the granularity or reorganizing the taxonomy - to name just two examples - is not handled by these systems. However, from observing users of a knowledge acquisition system, it becomes quite obvious that very often the user first chooses inappropriate representations. For instance, the decision whether a concept should be represented as an object (or event), as an attribute, or as a value determines further expressability of facts. Revising such a decision should be supported by the system. Only then can it be called a system supporting modeling.
This point will be discussed in more detail in section 2.4. Here, it suffices to point out that even advanced knowledge acquisition systems still fit into the transfer as opposed to the modeling view.

1.2 The Expert as a "Bottle"

Another way to interpret the bottleneck metaphor is to identify the expert with the bottle. The problem of knowledge acquisition is then to elicit the knowledge from the expert. Knowledge elicitation techniques such as interview techniques, content and protocol analysis, and the application of the personal construct theory (Kelly 55) are psychological means to this end. The way from the model supposedly in the expert's head of the expert to a knowledge base is divided into two steps: the elicitation and the encoding step. Often it is argued in favor of a mediating or intermediate representation between both steps, (e.g. Johnson 87; Young, Gammack 87).

How far the first step can be supported by a system is an open question. Knowledge elicitation primarily deals with the interaction of expert and knowledge engineer. This interaction requires social and communicative skills which cannot be ascribed to computers today. Manual knowledge elicitation

therefore still remains the most broadly used method for the first step of knowledge acquisition (LaFrance 87).

Knowledge acquisition systems stressing the elicitation step are, e.g. AQUINAS (Boose, Bradshaw 87), KSSO (Gaines 87), and KRITON (Diederich, Ruhmann, May 87). These systems interview the expert or knowledge engineer and form a representation of the knowledge which can then be transferred to an expert system. The user is guided through an acquisition dialog, in which s/he enumerates the objects of the domain, their relations and properties. The interfaces are so constructed as to please the user, thus compensating for lacking human communicative abilities.

Knowledge elicitation techniques take into account interview effects such as the influence of the form of a question to the answer. LaFrance (87) points out that asking the expert questions is like a search: the answer is in the mind of the expert but it might be missed by a question following the wrong path. This clearly shows that knowledge elicitation also presupposes the model itself to be fixed and complete in the head of the expert. Therefore, modeling is taken to be accomplished before the acquisition starts, and knowledge elicitation systems do not support the modeling phase.

2 Knowledge Acquisition as Modeling

In this section we want to inspect the modeling process and discuss some properties of the model which the expert has in mind. Is the model already completed in the head of the expert or is knowledge acquisition (including knowledge elicitation) the process of creating that model interactively? First, we argue that expertise need not rely on a model. Therefore, a good task-performer need not have a good model of the domain which could be elicited. Second, we describe the modeling process as scientific investigation. Psychological and sociological theories cast light on the nature of the modeling process. They also point out the interactive nature of modeling, which is the third point to be discussed. Finally, we derive from the discussion of models and the modeling process a view on knowledge acquisition as modeling. In order to make clear what is required of a system supporting modeling, modeling is contrasted with stepwise refining knowledge acquisition.

2.1 Knowledge vs. Skill

Knowledge is explicable and more or less conscious [2]. Skill is not explicable or conscious. This short characterization points to different types of expertise: one based on knowledge, and one based on skills. Good task performance need not rely on knowledge but may well rely on skills. Take for an example the arts. Ask composers, artists, poets how they came to create the composition, the picture, the poem - most often you will get a better answer from a critic. Where the artist is the expert in the sense of the person who is able to perform a task perfectly, the critic is the expert in the sense that he has a theory

[2] "More or less" because the implications of knowledge are not always conscious, too, and one knows not always what one knows.

explaining the result of the task performance. This distinction is important because elicitation techniques are only applicable to interviewing the theoretician, not to interviewing the task-performer.

Another example will make this point evident. Let us take natural language as the domain. Native speakers are the experts for their mothertongue in the sense of task-performers. Their linguistic competence is their skill. In a language knowledge base we want to have (at least) a grammar, a lexicon, semantic rules, word meanings, and morphological information. Now, imagine how manual elicitation as the one presented by LaFrance look like if we apply it to this domain.

> "Could you describe the kinds of things that native speakers do?"
>
> "Are there different types of native speakers? Is speaking a subtype of some other kind of communication?"
>
> "You said that thinking about what to say occurs before speaking. Why is that the case?"
>
> "Let me play devil's advocate. What if you were thinking while speaking?"
>
> "What are the basic objects in natural language?"

All these questions raise points of great theoretical interest: the first question could be the starting point for discussing speech act theory or communication and action theory; the second in addition points to variation in language; the third and fourth are heatedly discussed in natural language generation; the last question can be viewed as the root of all linguistic research for centuries. Of course, the native speaker is not aware of the theoretical implications of the questions. Before being interviewed, he may well have never thought about it. So he just starts with developing explanations for his own behavior, probably using what he has learned in school, some general explanation patterns, and introspection. Analogous to "naive physics", the model which people produce if they have to explain their expertise in dealing with the physical environment efficiently, a "naive linguistics" is developed. It is merely a rhetorical question, whether we want to build a knowledge base with naive linguistics for a knowledge-based natural language system. We surely do not! This example shows that skills cannot be elicited by the methods of knowledge elicitation and that the best task-performer is not necessarily the right person interviewed if we want to get knowledge about a domain. What the good task-performer (here: the native speaker) is good at is providing and assessing examples of task-performance (here: example sentences).

Figure 2 shows the relation between knowledge and skill with respect to knowledge-based systems. The important point is that people's skills are *described* for knowledge-based systems. Describing competence is a scientific task. In our linguistic example, describing adequately the linguistic competence can viewed as the central task of linguistics. We call the process of describing competence **modeling.** For knowledge-based systems, we demand, in addition, that the model must be operational, i.e. the system should be able to produce good taskperformance with the help of the model. In our above example, generative linguistics satifies this requirement: the grammar is not only descriptive but, at the same time, can produce or analyze sentences. A **model** is an explicit, explainable, and operational theory of a domain. The two lines of Artificial Intelligence research, the more application oriented and

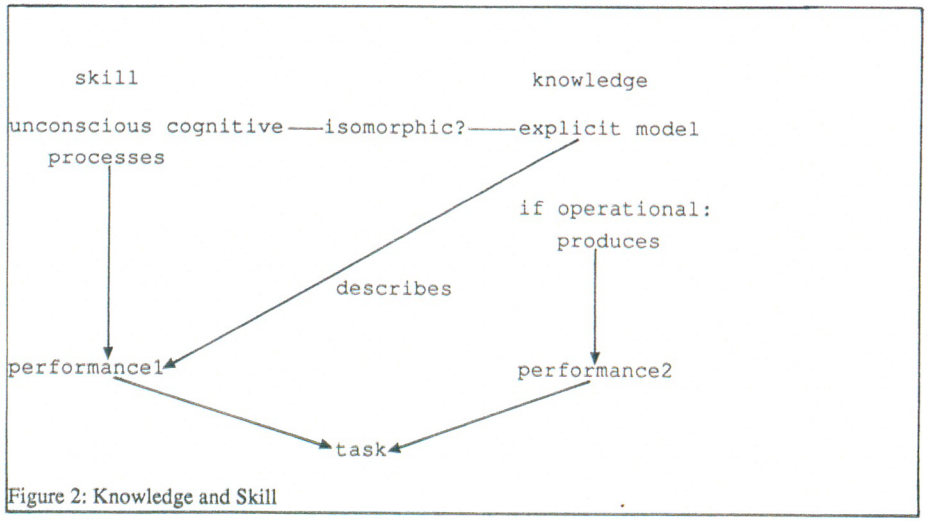


Figure 2: Knowledge and Skill

the cognitive oriented one, correspond to whether the model is claimed to be an isomorphic representation of the human competence and performance (Newell, Simon 72) or to whether it is sufficient that the model gives a good performance (Waterman 86).

2.2 Scientific vs. Naive Theories

The native speaker example in the section above already indicated that modeling is a scientific process. This does not exclude common-sense theories. The same cognitive processes are involved in building scientific and every-day life theories. As phenomenologists and symbolic interactionists have shown, constructing a model of reality is always embedded into the context of every-day life and its social reality (Schütz 62:256). The activity of producing knowledge which selects and explains the relevant observations, thus providing a basis for social interaction, is the same for common sense reasoning as for scientific reasoning (Garfinkel 67:279). Both kinds of theories are always provisional, although both are taken for granted in order to elaborate the theory. The main difference between every-day life and scientific theories is that scientific theories are systematically exposed to doubt and counter-examples. Institutions such as conferences with their refereeing procedure for acceptance and the discussions after talks guarantee that counter-examples or arguments against a theory are brought up and lead to a revision of the theory built up so far. This makes scientific theories more robust with respect to unforeseen events.

If we want to know how people normally build models, we may also look at naive psychology and its scientific investigation. In the field of perception of other people and in the field of attribute theory (derived from balance or dissonance theory) we find interesting results concerning introspection: the self-image is made up by the very same attribution process as is the image of other people. Herkner (80:46ff) reports a number of experiments which give evidence for this thesis. We learn from this that there is no direct link to inner processes of the self. Therefore, experts produce an explanation of their

own behavior just in the same way as they would produce it for other people with that behavior [3]. If an expert is expert at performing a certain task and not at producing explanations for this task-performance, s/he will produce a naive theory. The naive theory - as it is not tested by counter-examples and systematic doubt - is just the first approach to modeling. The danger for knowledge acquisition following from the sociological and psychological theories just alluded to is that we build systems with "naive knowledge bases" and confuse this with a knowledge acquisition problem!

2.3 Interactive Nature of Modeling

It has often been reported that experts changed their own thinking after working with a knowledge engineer (e.g. Turkle 84). They developed a model of their own expertise. Moreover, they adapted to the computer model which became their conscious model of what they were doing. This again gives evidence that the acquired model was not in the head of the expert before the acquisition process started, but was built up during the knowledge acquisition.

Modeling is driven by a need for explanation. Facts which are normally overseen become an object of attention if a crisis or conflict occurs or if the person starts thinking - regardless of why - with a theoretical attitude (Garfinkel 73). Interviewing somebody can well be regarded as engendering a theoretical attitude. Note also the stress LaFrance (87) is laying on playing the devil's advocate in interviewing: confronting the evolving model with counter-examples enhances its quality. In science, we can presuppose the theoretical attitude. But even there, the interaction between the scientists drives the scientific evolution. The commonly agreed upon assumptions underlying a theory as well as objections against the theory are a matter of interactive agreement. Taking this statement of phenomenologists and interactionsists seriously we also regard the process of knowledge acquisition as an interactive one. The model of the domain is not already in the head of the expert but is interactively constructed by the expert and the knowledge engineer. This view corresponds exactly to the observation mentioned above.

2.4 The Modeling Cycle

Let us now summarize what we have learned about models and the process of making them.

1. Good task-performance need not rely on an explicable model of the domain.
2. The expert who is a good task-performer is not necessarily also an expert who knows about the domain.
3. A model can be an operational one.
4. Modeling is a scientific process.
5. Interviewing task-performers may start their modeling activity.

---

[3] The main difference between the self-image and the image of other people lies in the parameter "distance": the self is always close to itself, while other people can be distant or close. Closeness gives a positive bias to the explanation forming.

6. A model which is not exposed to counter-examples and doubt sytematically and over a longer period of time is a naive theory.
7. Modeling is an interactive process.

Applying these points to knowledge acquisition we get the following picture:

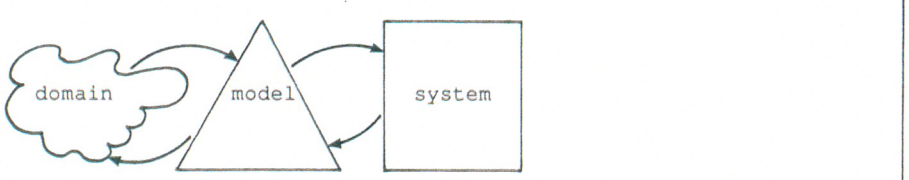


Figure 3: Knowledge acquisition as modeling

The important difference as compared to the transfer view is that the feedback is present from the system to the model-building activity and from the model to observations or experiments in the domain. That is, modeling - as science - is a cyclical, not linear process.

A closer look at the modeling cycle reveals three phases and three types of revisions. This cycle is independent of whether computers are involved or not. It corresponds to the cycle in scientific research. In the first phase, the framework of the model is laid out. It is determined which aspects of the domain are relevant and are to be included in the model, the vocabulary for describing phenomena of the domain is chosen, and some semantic relations between concepts, properties, or states are taken as basic assumptions. Normally, we write this framework down on paper before we choose an appropriate computational representation for it. The step from the domain to the model, especially the creation of the first tentative representation, is not system-supported.

In the second phase, facts and rules are filled into the framework. By working out the framework, it becomes finer grained and more complex. More and more semantic relations are established, and more and more observations of the domain are represented. If there are observations in the domain which are considered to be important but cannot be reflected within the framework, the framework has to be revised. Due to the great difficulty of this revision the step from the domain to the model is normally not system supported. The framework is the basis on which the model is built. Changing the basis, of course, effects changes in the model. In fact, desired changes of the model are the reason for changing the representational framework. The problem is to formulate the effects in a language which itself is subject to change. People, however, seem to be able to reformulate, reorganize, restructure, and refine a model - and systems should be able to support them in doing it! In the next section, an example for this kind of revision is discussed, indicating the requirements of a system supporting the revision.

In the third phase, the model built-up so far is evaluated. In order
- to test the model,
- to clarify conflicts which cannot be resolved with the given information, and
- to fill in underdeveloped areas of the model

experiments are designed. Experiments make new observations available. The new facts may then be used to validate the model.

Evaluating the model implies revisions of facts and rules. This type of revision can be supported by a system, today [4]. Consequences of retracting a fact are determined and the consistent state of the knowledge is maintained.

However, the third phase involves also another revision. It may turn out that some basic assumptions have to be retracted which, in turn, invalidates a lot of facts and rules. If we do not want an empty model again, facts and rules should be adjusted to new basic assumptions [5].

An intermediate representation on paper of the model, which is then encoded in a computational representation language, is necessary if the system which maintains the knowledge cannot handle the three revisions of the modeling cycle. The revisions are then performed on the model, and the encoding is undertaken when no major revisions are expected. In contrast, a system supported modeling cycle integrates intermediate representations into the system (see figure 4). The challenge for a system supporting the modeling cycle lies in the reversability of facts and rules as well as terminology and basic assumption.

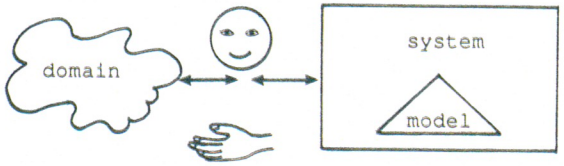


Figure 4: System supporting the modeling cycle

Because today's hand-eye technology does not allow for direct observations and experiments in a domain, a person mediates domain and system, but this is not relevant for our concern. The important point is that the person builds up and revises the model from the first phase to the third and then entering the first phase again with the help of the system.

2.5 Modeling vs. Stepwise refinement

In this section I discuss two examples in order to contrast the system support for modeling with the support today's knowledge representation (acquisition) systems offer. Today's knowledge representation systems acquire knowledge in a stepwise restricting manner: each definition restricts the space of further

---

[4] Of course, belief revision is also not a solved and fixed problem!

[5] Emde (87a) presents a learning program, METAXA.3, which re-classifies facts because of theory revision.

possible definitions, facts, and rules. It is impossible to retract a definition because of incoming new definitions or - even worse - new facts. Instead, the conflicting new definitions or facts are rejected. The advantage of stepwise restriction is the sound logical basis. The disadvantage is that it forces the user to a top-down procedure in acquisition: defining terminology from the most general to more specific terms, then giving rules and facts which are controled by the terminology [6]. This procedure is adequate in some cases, namely when a well defined terminology already exists. In situations, however, where the user develops the terminology with respect to facts, this procedure is not helpful. In particular for revision, a system-supported way back from the more specific entries to the more general ones is missing. In practice, the user first models the domain on paper or with a text editor. There, the user is allowed to proceed from the more specific to the general as well as from the general to the more specific. The user revises the evolving model by hand until the terminology is almost completed. Then, the user enters the domain model into the acquisition system starting with the terminology. If additional revisions are necessary, the knowledge base is loaded into an editor, revisions are made by hand, and the edited knowledge base is loaded back into the system. Thus, the user is responsible for all effects of the changes because a text editor has no ability of revision and maintenance.

Typical examples for stepwize restricting knowledge acquisition are KL-ONE-like systems (Brachman, Schmolze 85). Let us look at an example of upwards revision in the terminology. Figure 5 shows an excerpt of a T-Box.

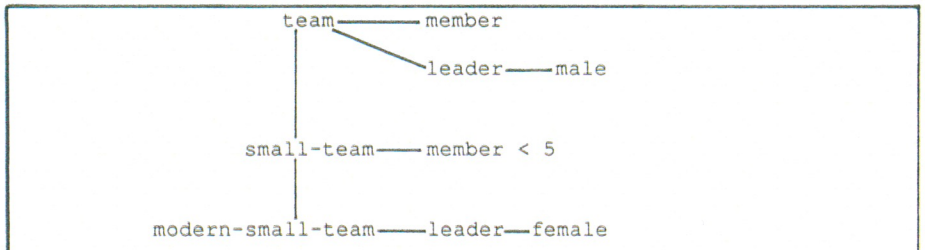


Figure 5: Excerpt of a T-Box

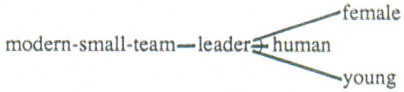
A *team* is a primitive concept with the roles *member* and *leader*. The leader is assumed to be *male*. A *small-team* is defined as a *team* with at most 4 *members*. Let there be the rule that *small-teams* which are guided by a woman get a special grant from an equal-rights-fund. We now want to define a *modern-small-team* as a *small-team* with a *female leader*. This new definition would be rejected by stepwise restricting acquisition systems because it contradicts the definition of the superconcept *team*. However, it is often the case that we are very convinced of a definition of a more particular concept and want to change the superconcept definitions accordingly. This is quite natural, since we cannot think of all consequences beforehand and therefore often take default definitions for general concepts. These

[6] Terminology controls facts and rules in that it defines the semantics of the objects, attributes, and relations which are used to write facts and rules.

definitions must be overwritable when more specific definitions come in. Of course, there are several possibilities to change the T-Box as to allow the definition of the *modern-small-team*. We could retract the superconcept-relation between *modern-small-team* and *small-team*, create a new concept *female-guided-team* and put *modern-small-team* below it. This, however, would require doubling the restriction of the number of *member*. Less changes are necessary if we change the restriction of *leader* in the concept *team* to *human*. Note, that changing the restriction to *female* would affect the definition of *small-team* in an undesired way: it would become the same concept as *modern-small-team* and no *small-teams* with a male leader would remain. This rather simple example shows what a system must be capable of in order to support modeling. It must detect contradictions, find possible ways to resolve the contradiction, compute the consequences of the resolution, and choose the possibility with the least changes.

To make things even more complicated, let us look at a second example. Here, a definition is to be changed because of facts. Let us imagine a typical bureaucratic situation: it is known that grants are given to *modern-small-teams*, and we thought that *modern-small-teams* are always guided by a woman. Now, however, we hear of a team which got the grant but is guided by a man. We know the facts, and want to adjust our definitions to them. Figure 6 shows an excerpt of a T-Box and an A-Box.

T-Box



A-Box

leader(x,y) modern-small-team (x) young (y) male(y)
leader(a,b) modern-small-team (a) young (b) female(b)
leader(c,d) modern-small-team (c) young (c) male (c)
leader(e,f) modern-small-team (e) not (old(f))

Figure 6: Excerpt from a T- and a A-Box

In order to be able to enter the team x and the team c, we can change the definitions in the T-Box in different ways. If we regard all the teams which got the grant, we find that they have in common that the leader is young or at least not old. Changing the definition accordingly requires not only the change of the particular value *female* into *human* but also the introduction of another role, namely *age*. Changing the definition in this **bottom-up** way is, in fact, a **learning** step. Because observations are what is most certain, definitions should be reversable - we cannot change the observations to fit to our definitions! This demands, however, for the definition of revising operations, the computation of all consequences and a selection algorithm which chooses the best change. Before this is achieved, no system can fully support the modeling cycle developed in a top-down as well as a bottom-up manner, but rather must stick to the stepwise refining paradigm. This is to say that **modeling** should not be taken as a new word

for an old thing but to indicate the work that has to be done in order to support modeling.

3 Machine Learning as Automatic Modeling

In the field of Machine Learning, many aspects of modeling have been investigated, and prototype systems which perform some tasks of the modeling cycle have already been built. In this section, we give some examples of machine learning programs which perform part of the modeling task. Viewing machine learning as automatic modeling and knowledge acquisition as a modeling task, it is an obvious idea to integrate machine learning into knowledge acquisition. A new way of integration is then exemplified by the system BLIP.

3.1 Aspects of Modeling in Machine Learning

It would take too much space to report on all the efforts to automate parts of the modeling cycle or on all the work on theory formation (which can be viewed as another word for model-building) which has been undertaken in the field of machine learning. However, pointing to some example system may make clear that, in fact, machine learning is an attempt to operationalize modeling.

Finding the concepts with their defining properties and deleting irrelevant parts of object descriptions corresponds to an early step in the first phase of modeling. Having a system perform this task has long been a goal of machine learning. Going further, taxonomies of concepts have been automatically built up. Techniques for concept formation and learning class descriptions have been developed by Michalski (Michalski 80; Michalski, Chilausky 80; Michalski,Stepp 83), Winston (75), and Mitchell (82) - to name but a few. Learning classification trees (or rules) which can be regarded as concept formation from another point of view is also a task of the first modeling phase (Michie, Bratko 78; Quinlan 86).

Discovering regularities, generating hypotheses, testing them, and using them to create a knowledge base with a considerable coverage is performed - by very different techniques - by systems such as INDUCE (Dietterich,Michalski 81), SPARC/E (Dietterich,Michalski 85), BACON (Langley, Bradshaw, Simon 83) and META-DENDRAL (Buchanan,Mitchell 78). Here, part of the second phase of the modeling cycle is automated. Discovering laws (Lenat 77, Langley 81) is, probably, the most evident correspondent of machine learning to the development in science. In the scientific process, the theory built up so far is used to guide further research. Analogeously, a theory-driven approach has now been followed by discovery programs (Michalski, Falkenhainer 87, Kokar 87).

Enhancing problem-solving performance by chunking operators (Laird, Rosenbloom,Newell 86) or creating a knowledge structure from primitive elements (Pazzani 87) is an aspect of re-representing in modeling.

Refining rules because of conflicts (Michalski 85, Wrobel 87c) or failures (Schank 82; Rajamoney, DeJong, Faltings 85), or master's advice (Mitchell, Mahadevan, Steinberg 85, Kodratoff, Tecuci 87b) are other tasks out of the second modeling phase which are operationalized by machine learning programs.

Finding explanations for observations has become a hot topic in machine learning (Rajamoney,DeJong,Faltings 87; Wilkens 86; Pazzani 86). Of course, all rules or laws found by a program should explain the facts and predict new facts. Thus, just as science is always creating explanations, so do learning programs (cf. Kodratoff 86). However, at an early stage of the scientific proces one is bound to empirical discovery, whereas with a better developed model it becomes possible to learn a lot from just one example.

Taking remembered solutions as a basis for a new model to be built or for a problem to be solved is dealt with in learning by analogy (Carbonell 83).

The third modeling phase deals with theory as such, as opposed to single concepts or rules. Forming a theory (Amarel 86) as well as revising it (Emde 87a, Rose, Langley 86) is part of the third phase. Validating the model and gathering new data by conducting experiments is another part of this phase. In the field of knowledge acquisition and machine learning, some attempts are now being made to design experiments automatically (Wisniewski,Winston,Smith,Kleyn 86; Rajamoney,Dejong, Faltings 87; Dietterich 86).

As opposed to stepwise restricting acquisition systems, learning systems proceed from extensions to the intension of sets. Definitions are built up incrementally and can be changed and enhanced baised on new facts. This well suits the demand for reversibility. However, machine learning also faces the problem of changing the representation which was taken as a starting point - yet at another level. The examples - whether selected by a teacher or coded observations - have to be represented so that the learning program can use them to form concepts, taxonomies, or definitions, which in turn can be used to express more facts and rules about the domain. Choosing the appropriate description language for the examples determines the possible outcome of the learning program. Revising the description language because of unsatisfactory learning results is a most challenging approach for meeting the reversibility requirement (Amarel 68). Today, this task is left either to a teacher in most similarity-based learning programs, or to the knowledge engineer who has built the knowledge base which is refined by explanation-based learning.

3.2 Integrating Machine Learning into Knowledge Acquisition

Machine learning has been successfully applied to building large parts of knowledge bases. Examples of such successful learning programs are ID3 (Quinlan 83) and AQ11 or INDUCE (Michalski 80). Most often, the learning system and the performance system are linked **serially:** the learning system acquires rules independently from the knowledge which is already stored in the expert system. It delivers rules (or facts, or tables) to the performance system without getting or using any feedback from it. Another way of putting learning programs to good use in knowledge acquisition is shown by apprenticeship-learning, where the learning program **interacts** with the expert system. It compares the performance system's solution with a solution given by an expert, explains the difference, and changes the knowledge base accordingly. An example of such a system is LEAP (Mitchell, Mahadevan, Steinberg 85). Together with the expert system VEXED, the architecture of the system shows that both system and user work on the same knowledge, in this case a rule (see figure 7).

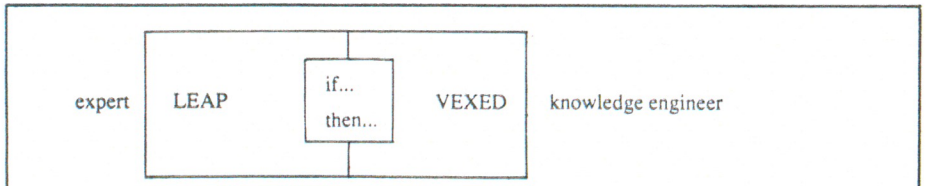


Figure 7: The architecture of LEAP

However, LEAP is restricted to the end of the second phase of the modeling cycle.

Another interactive knowledge acquisition system of an architecture similar to LEAP's is DISCIPLE (Kodratoff, Tecuci 87b). As does LEAP, DISCIPLE learns from one rule given by the user and, therefore, needs a verification for the step from this one example to a general rule. Verification in DISCIPLE is explanation. In contrast to LEAP, DISCIPLE does not presuppose a strong domain theory. Instead, the system integrates the user into the verification process: it selects the possibly interesting explanations for a rule, forms necessary conditions for the rule, and generates examples for the generalized application condition of that rule, each time asking the user to classify the system's results. The outcome of the learning process is a qualification for a rule which can be a disjunction. However, also DISCIPLE relies heavily on the quality of the domain theory. Whether explanations can be found and whether they are interesting depends on the expertise of the knowledge engineer who encodes objects, properties, and relations. The material for an explanation is a pattern of - possibly indirect - relations among the objects involved in the user's rule. Part of the generalization of the applicability condition is performed with the help of theorems which lead from a specific relation to a more general one. The domain theory may be weak, but it must have a rich structure and carefully constructed theorems giving a hierarchy of relations. The construction of the domain theory itself is not system-supported. The outcome of the learning program is not fed back into the domain theory.

A third way to integrate machine learning into knowledge acquisition is **balanced cooperative modeling.** A knowledge-acquisition system supports the user in modeling and, at the same time, enhances this model by learning. The user is the person who builds up the domain model, the knowledge engineer. It is not a person showing skills in problem solving, but a person providing the system with *knowledge* - and thus perhaps showing some skills in describing phenomena. The role of the user is that of a scientist who writes down observations, structures them, and finds rules which cover the phenomena. The role of the system is that of an assistant looking over the user's shoulder, compiling information, taking care of the book-keeping, and cleaning-up consequences of the user's changes, pointing to hidden conflicts, and recommending enhancements for the model. The enhancements include new rules induced from the user's notices, new concepts, and structuring of the objects into classes. Of course, the user is also free to reject the recommendations of the system. Figure 8 shows the architecture of such a system.

A consequence of this approach is that the learning part of the system has to cope with an incomplete and probably wrong domain theory because building up the theory and learning from it takes place at the same time. Another implication of this is that the domain model must be reversible at all points because

either the user or the system can make a mistake. BLIP is an example of the balanced cooperative modeling approach. We call this approach "sloppy modeling" in order to point out that the user may start with a very tentative (sloppy) layout of the model and then change and enhance it with the help of the system.

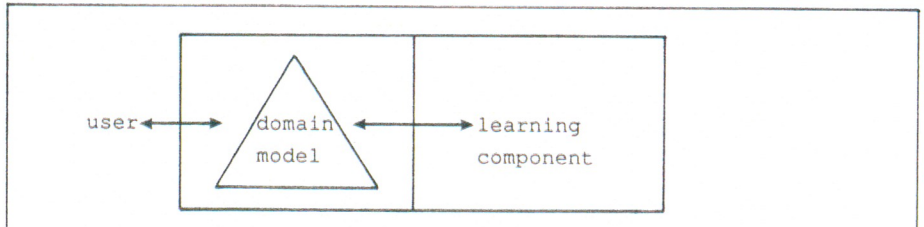


Figure 8: Architecture of balanced cooperative modeling

3.3 The BLIP System

BLIP is a system which supports the modeling activity of the user through the first two phases by a convenient interface, a coordinator of different knowledge sources, and by watching consistency and integrity. The user may easily follow the bottom-up procedure or the top-down procedure of acquisition. Terminological knowledge can also be changed; the system manages the consequences of the revisions. BLIP is also an active modeling system which is designed to enhance the model by learning and revising. Moreover, the same procedures which integrate the new facts, rules, or objects given by the user also incorporate those given by another part of the system, the learning component.

Relating BLIP to our results about modeling (cf. 2.4), an important point is that BLIP is not a skill acquisition system, nor does it acquire knowledge from observing skilled task-performance, nor does it suppose the user is a good task-performer. The BLIP user is supposed to deal with explicable knowledge. However, this explicable knowledge need not be the rules (laws) of the domain; it can also be observed, factual knowledge. The modeling activity of user and system is regarded as a scientific process, as contrasted to the training process of expertise.

The question of whether the model built up by BLIP is an operational one needs a little more explanation. BLIP is *not* a consultative system or integrated into a consultative system. Of course, deductions can be made by BLIP's inference engine. Viewing this as problem solving (as is done, e.g. for the DISCIPLE system), the user can solve problems using BLIP. In this sense, the domain model of BLIP is an operational one. However, for a consultative system, many more capabilities are required: an efficient and explicit strategy for asking the user for the specification of his problem (cf. the MYCIN rule with the implicit age limit quoted above), an explicitly represented strategy for problem solving (cf. the MYCIN heuristic quoted above), and explanation capabilities. None of these capabilities are present in BLIP. In the sense of a consultative system, BLIP's domain model is not operational.

BLIP does not generate possible counter-examples in order to present them to the user (as is done by DISCIPLE). Thus, it is the user's task to test and verify the evolving domain model. The third phase of

the modeling cycle is not supported by BLIP.

The insight that modeling is an interactive process was the guideline for the system design of BLIP. The terminal screen can be viewed as the scratch-pad for system and user. The important point here is that the system really processes on the epistemic or ontological level the inputs and changes of the learning part and the user with all consequences - as opposed to an editor which only handles strings.

In this section, an overview of the BLIP system is presented. We then concentrate on revising possibilities in BLIP and show some examples.

3.3.1 Knowledge Representation of BLIP

Knowledge in BLIP is represented by **facts** and **rules.** Facts are expressed by predicates with constants as arguments. Syntactically, predicates are declared to have arguments of a certain **sort.** Each constant belongs at least to one sort. Rules may include negation and conjunction and have only one conclusion. Rules whose predicate's arguments are variables are written as **meta-facts.** Thus, meta-knowledge in BLIP has nothing to do with strategic knowledge, but is knowledge about the domain itself. It is a declarative representation for rules. Meta-facts are expressed by **meta-predicates** with predicates as arguments. Meta-predicates are defined by associating the predicate with a rule-schema.

The step from a meta-fact to a rule is performed by a rule-generating rule which uses the meta-predicate's declaration (cf. Emde 87b). The declarative representation of rules as meta-facts provides for reasoning about rules, checking the consistency of rules, and deriving rules from other rules. The means for that are **meta-rules.** Meta-rules are rules with predicates as arguments of higher-order predicates. Also meta-rules can be represented redundantly as facts, namely meta-meta-facts (cf. Thieme 87). The type restriction (the predicate must always be of a higher type than its arguments) is obeyed.

BLIP uses higher-order concepts. It does not realize, however, a full higher-order logic because that would result in unacceptable computational properties of the knowledge representation. As Wrobel (87b) has shown, BLIP's representation language is fact-complete and allows for answering any query within a time that is polynomial to the size of the knowledge base.

3.3.2 BLIP's Architecture

The main components of BLIP are the **coordinator,** the **modeler,** the **inference engine,** and the **interface.** The components interact with each other by sending and receiving notifications. This is the reason why the user's input is treated in the same way and handled by the same processes as the "input" of the modeler: the notifications sent to the coordinator are of the same form. The coordinator is the module which maintains the integrity of diverse knowledge sources. It can be viewed as the environment of the knowledge sources which represent the domain model (Morik 87). It also includes a program which acquires and (re-)organizes the sorts and syntactic declarations of predicates and maintains the membership of constants to sorts (cf. example below). The modeler is the component which evolves the domain model automatically. It includes a program which acquires meta-knowledge (cf. Thieme 87),

a program which learns new rules, and a program which forms concepts (Wrobel 87c). The inference engine stores facts and rules from the object level as well as those from meta-levels (Emde 87b).

Rule-learning in BLIP is, in fact, learning meta-facts. The procedure is based on experience with the learning system METAXA (Emde 84). Hypotheses of possible meta-facts are generated and tested against the facts known so far to the system. The space of hypotheses is limited by syntactical restrictions which are attached to meta-predicates. These restrictions concern the argument sorts of predicates of a rule. Heuristics select the hypotheses to be actually generated and tested. Hypotheses are tested with the help of **characteristic situations.** Characteristic situations are search patterns which collect verifying and falsifying facts for a rule-schema. Verified hypotheses become known meta-facts and the rule-generating rule enters the corresponding rule into the inference engine. The rule, in turn, is used to derive new facts. In this way, induction and deduction work together to enlarge the domain model. BLIP is a closed-loop learning system.

The interface reacts immediately, showing the user all consequences of an action - be it performed by the user or by the modeler. Compared to the edit-and-compile behavior of many systems, this immediate feedback is an advantage for the user. The system can quickly be learned, a mental model of the system can be built very easily and naturally, and the processes become transparent and inspectable. For more details about the interface see Wrobel(87a).

3.3.3 Modeling with BLIP

In general, modeling with BLIP means interactively defining concepts and finding rules. Inference rules in BLIP can be terminological rules (those which express the semantics of a concept or relation) or empirical rules (those which describe or summarize a set of facts). When starting to build up a domain model with BLIP, the system contains nothing but meta-knowledge (the user can choose among several sets of meta-knowledge). The user then has to declare the predicates s/he wants to use in order to represent facts and rules. This is the very first phase of defining a terminology. The following part is to establish the semantics of the predicates by finding meta-facts concerning them. BLIP does not support the third phase of the modeling cycle: no experiments for evaluating alternative domain models are designed. However, the first and the second phase being fully reversible, the reorganization of a model needs no particular phase in its own right but is integrated into the first and second modeling phase.

In the following, we discuss how BLIP supports the first and second phase of the modeling cycle with a specific focus on the reversibility options.

3.3.3.1 Layout of representational framework

In this section, the interactive layout of the representational framework for a domain model is described. Four typical types of inappropriate modeling are differentiated, and it is shown how BLIP helps to recover from inadequate first modeling attempts,. thus illustrating the sloppy modeling approach.

BLIP supports the declaration of the predicates and - if needed - the introduction of additional

meta-(meta-)predicates. The declarations can be performed in a bottom-up manner and are fully reversable. The system support for declaring meta-(meta-)representations is described in Thieme (87). The system support for predicate definitions is based on collecting all constants together with their membership to a sort. Thus, if a predicate is declared and facts are inputted, a new predicate with (some of) these sorts need not be declared, but rather corresponding facts with known constants can immediately be entered. The predicate declaration for the new predicate is then automatically built. The more important use of the collected constants is the automatic and reversible arrangement of the sorts. Sorts with their intersections are transferred into classes which are arranged in a lattice. With more and more facts, the intersections and subset and superset relations between sorts can change, and, accordingly, the lattice of classes is revised. Thus, also in this respect, the user may largely follow a bottom-up routine (i.e. entering facts) and the system signs responsible for necessary revisions (Kietz 88). Further work on sorts will deal with sort predicates; if for each sort there is a corresponding predicate, the granularity of the model can be changed from objects taken as sorts to objects taken as predicates. Furthermore, the meta-facts about sort-predicates can then be learned.

With the syntactic declaration of predicates, the user decides which entities are to be represented as objects, as object classes, as predicates, or as rules. This choice determines the adequacy of the model with respect to its further use. It also determines the applicability of BLIP's learning. BLIP learns about predicates. Thus, the most interesting things should be represented as predicates. However, it may well happen that the user first chooses an inappropriate representation. Our demand for reversibility means, in this context, that the system should be able to do something about it.

What are inappropriate first modeling attempts and how does BLIP support the recovery from them? We want to discuss four widespread problems of laying out the representational framework, thus illustrating the BLIP approach. As an example domain, let us take hepatitis. To make the example simple, we just want to express that a yellow skin color indicates that the liver is disturbed in its function. The first attempt of a user to represent this could be writing the following rule:
color (*yellow*) -> *disturbed* (*liver*)
In this attempt, no general, i.e. all quantified, rule is possible:
color (*x*) -> *disturbed* (*z*)
does not make any sense, because z is "unbound". All that is interesting is expressed by constant terms. We call this type of inadequate modeling the **case of the unbound variable.** The user recognizing the inappropriateness of this representation might change it by introducing a new predicate
relation (<*color*>, <*organ*>)
and change the rule into
relation (*x*,*z*) & *color* (*x*) -> *disturbed* (*z*).
In this case, no revision of accepted entries into the fact base are to be made. Some new facts can be entered relating the color of the skin to disfunctions of organs.

However, this is not a clever way to model the subject, either. One of the problems with this attempt is the **case of the missing argument.** All facts which could be entered in this framework would implicitly refer to the same patient who can only be yellow, pale, red, etc. Thus, different cases cannot be represented. A quantification over a set of patients is needed. This, in turn, requires the predicates *color*

and *disturbed* to have one more argument. We could then write
relation (*x*,*z*) & *color2* (*x*,*y*) -> *disturbed2* (*z*,*y*)
and enter facts such as
color2 (*yellow*, *tim*), *color2* (*pale*, *tom*), *disturbed2* (*liver*, *tim*).
In order to introduce the new two-place predicates, we could write
color (*x*) & *patient* (*y*) -> *color2* (*x*,*y*)
disturbed (*x*) & *patient* (*y*) -> *disturbed2* (*x*,*y*)
and give the name of the one (former implicit) patient by
patient (*tim*).
But this rule is not appropriate in general because it links any patient and any symptom. It only serves as a transformation of a fact base which models implicitly one patient's case. Such a transformation should be performed by changing the predicate's definition. Now, however, editing the predicate definition of *color* makes the already entered facts syntactically ill-formed, and there is no way for the system to insert the appropriate term into the second place of the predicate. Thus, the only solution here is to put a dummy symbol into the second place and leave it to the user to replace it.

The model developed so far expresses the interesting concepts *yellow* and *liver-disturbed* as constant terms. Thus, BLIP cannot learn about these concepts. This is an example of the **case of the interesting constants.** Using BLIP is much more effective if we represent the central concepts of a domain as predicates. Here, these predicates are
yellow (<*patient*>)
and
liver-disturbed (<*patient*>).
The symptom-to-disease relation is then expressed by the rule
yellow (*x*) -> *liver-disturbed* (*x*).
This representation makes the predicate *relation* superfluous which, in fact, is quite specialized since there are only a few diseases recognizable by skin color. Expressing symptoms by predicates, we can have other symptoms besides skin color for other disfunctioning organs besides the liver. If we want to maintain the already given facts but change the representation we can write
color2 (*yellow*, *x*) -> *yellow* (*x*)
and
disturbed2 (*liver*, *x*) -> *liver-disturbed* (*x*).
Modeling can go on with the new predicates, learning about them is enabled, and analogous relations between symptoms and diseases fit into this framework.

The fourth case we want to discuss here is the **case of the missing predicate.** It is quite common for the user to forget to introduce a predicate which discriminates different cases. Perhaps the user did not even realize that there are different cases. So far, changes in the representational framework were performed by the user and supported by BLIP. However, we expect a learning system to change the representation autonomously if the facts give reason to do so. And, in fact, BLIP is capable of introducing new predicates. BLIP recognizes that a predicate is missing by exceptions to a rule which show that the rule applies in too many cases. The rule must thus be restricted by discriminating the cases where it should

apply from those where it should not. As opposed to DISCIPLE, where just one rule is qualified by an applicability restriction, BLIP exploits exceptions for forming concepts which are not only used to restrict the particular rule but are also available for all processes. In particular, relations between the new predicate and other predicates can then be learned. The process of concept formation is described in Wrobel (87c).

The user need not restart modeling from scratch after recognizing that the first attempt to lay out the representation was more adequate; as has been illustrated, the model evolves from the **sloppy** first attempt to the more appropriate one without a loss of the already typed-in facts. The first phase of modeling can be reentered at any point, i.e. it is fully reversible in the sense discussed above.

3.3.3.2 Elaboration of the framework

The second phase of the modeling cycle is performed with BLIP by entering facts and rules. BLIP deduces additional facts and induces additional rules. Here,too, we want to concentrate on reversibility. To illustrate revise with BLIP, we take the BACK example from above (cf. 2.5). First, we assume that the user modeled a situation corresponding to that of figure 5 and show how with the support of BLIP the user can overwrite the definition of the superconcept. Handling contradictions at the meta-level as well as at the object level is demonstrated. Then we present the more effective use of BLIP: entering the facts corresponding to figure 6, BLIP learns the appropriate rules.

In order to model the T-Box behavior of BACK, we need the meta-meta-predicates that realize inheritance of value-restrictions and the meta-predicates realizing the superconcept relation and the set of disjoint values concerning a role. The meta-level predicates can be presupposed by the user, i.e. they need not be entered by the user, but can be chosen loading the KL-ONE-METAPRED set. Note that the different sets of meta-level predicates model different system behavior at the inferential level. If the user does not want to have value-restrictions inherited because this is considered too restrictive, another meta-(meta-)predicate set should be chosen. The rule expressing the superconcept-relation is
inclusive(*p*,*q*): *p* (*x*) -> *q* (*x*)
The rule expressing disjoint concepts is
opposite(*p*,*q*): *p* (*x*) -> *not* (*q* (*x*))
where because of
symmetric (*opposite*)
also
opposite(*q*,*p*): *q* (*x*) -> *not* (*p* (*x*))
holds. By writing the meta-fact
inclusive (*modern-team*, *team*)
the user establishes *modern-team* as subconcept of *team* [7]. By writing the meta-fact
opposite (*male*, *female*)

[7] The number-restriction of the member role is not representable in BLIP.

the user expresses disjoint concepts. For value restrictions we introduce the meta-predicate
valr (*p*, *q*, *r*): *p* (*x*,*y*) & *q* (*y*) -> *r* (*x*)
and for another constellation of predicates
valr2 (*p*, *q*, *r*): *p* (*x*, *y*) & *q* (*x*) -> *r* (*y*).
The T-Box mechanism can then be described by two meta-rules:
mm1 (*inclusive*, *valr*): *inclusive* (*p*, *q*) & *valr* (*l*, *m*,*q*) -> *valr* (*l*, *m*, *p*)
mm2 (*inclusive*, *valr*, *opposite*): *inclusive* (*p*, *q*) & *valr* (*l*, *m*, *q*) & *opposite* (*m*, *n*) ->
not (*valr* (*l*, *n*, *p*)).
The first meta-rule inherits the value-restriction, the second ensures consistency between rules. By writing
R1;*valr* (*leader*, *male*, *team*): *leader* (*x*, *y*) & *male* (*y*) -> *team* (*x*)
and
R2;*valr2* (*leader*, *team*, *male*): *leader* (*x*, *y*) & *team* (*x*) -> *male* (*y*)
the user defines the concept *team* corresponding to the definition in the KL-ONE example above. If the user,now wants to enter
R3;*valr* (*leader*, *female*, *modern-team*): *leader* (*x*, *y*) & *female* (*y*) -> *modern-team* (*x*)
R4:*valr2* (*leader*,*modern-team*,*female*): *leader* (*x*, *y*) & *modern-team* (*x*) -> *female* (*y*)
we get a contradiction between meta-facts. BLIP therefore first rejects the entry and informs the user of the contradiction and of the possibility of entering the same contradictory meta-fact again. If the user then selects the same entry-menu as before, thus insisting on the second definition, it is stored in the inference engine as a contradictory rule. Up to now, the knowledge revision on the meta-level has to be done by the user; s/he has to decide which of the conflicting meta-facts is to be changed or deleted. If the user decides to delete the rule which was restricting teams to groups with male leaders, the facts derived from this rule are retracted. The user could also delete the responsible meta-rule so that the derived meta-fact
not (*valr* (*leader*, *female*, *modern-team*))
is retracted. The inference engine is capable of keeping track of the derivations at the object level as well as at the meta level (Emde 87b).

As meta-rules watch consistencies between rules, rules are watching consistency between facts. Thus, if the user wants to enter
modern-team (*kit-back*)
leader (*kit-back*, *kai*)
male (*kai*),
BLIP first refuses to store *male* (*kai*) and informs the user of the contradiction. But if the user is really convinced of the fact and inputs the fact again, knowledge revision starts in order to solve the conflict. Here, there is not enough known to revise autonomously. Therefore, BLIP asks the user whether s/he wants to provide a better support set for R4 or to delete the rule. If R4 is deleted and, as the user expects, all facts derived by R4 are retracted, we don't have a proper definition of *modern-team* any more. Of course, the user can give another defintion by typing in a meta-fact. However, the easier way is to input the facts and let BLIP find the appropriate value-restriction defining the concept *modern-team*. From the following facts

male (*kai*), *male* (*michael*), *female* (*christa*), *female* (*katharina*),
modern-team (*kit-back*), *modern-team* (*kit-natan*), *modern-team* (*kit-fast*), *modern-team* (*kit-lerner*),
young (*kai*), *young* (*michael*), *young* (*christa*), *young* (*katharina*),
leader (*kit-back*, *kai*), *leader* (*kit-natan*, *michael*), *leader* (*kit-fast*, *christa*), *leader* (*kit-lerner*, *katharina*).

BLIP learns - among other meta-facts -

valr (*leader*, *young*, *modern-team*)
and
valr2(*leader*, *modern-team*, *young*)

Thus, the best way to use BLIP for modeling is to enter facts, look at the results of BLIP's learning, correct them by giving negated facts or by editing the meta-facts or rules, and leave all the rest up to the system.

Meta-meta-facts, meta-facts, and facts can easily be retracted, and BLIP maintains consistency by retracting all **deduced** (meta-)facts. The problem of how to determine which meta-facts were **induced** from the retracted facts remains open. It would cost too much space to index and store the set of facts which instantiated a characteristic situation and thus helped to verify or (falsify) a hypothesis.

Modeling with BLIP is restricted to building up a domain model, corresponding to the first and the second phase of the modeling cycle. Because of revision possibilities, however, modeling need not be performed in sequential phases. Reorganization of the model, even of the representational framework, is always possible and is always system-supported. Thus, the infinite and cyclical nature of modeling is taken into account. Moreover, BLIP autonomously enriches the representation by forming new concepts and enhances the model by learning new rules. Thus, BLIP is an example for balanced cooperative modeling.

The user interacts with BLIP in different ways.
BLIP uses learned or given rules to deduce new facts, and if the user denies those facts, knowledge revision tries to modify the corresponding rules. The derived facts can be regarded as predictions, and the user judges whether they are accurate or not. Here, the user represents the 'real world' to the system.
Of course, the user, having all comforts of an inference engine, can ask BLIP questions about its knowledge. In this sense, a domain model in BLIP is an operational one.The user employs the system as a domain encyclopedia with inferential capabilities.
The typical user-system constellation, however, is that of a scientist and his assistant: a cooperative and interactive process of model-building guided by the scientist.

In addition to stepwise refinement, BLIP allows for a bottom-up procedure of modeling: starting with known facts and developing concepts and definitions on that basis. Changes in the fact base lead to changes of definitions and concepts.

4 Conclusion

If one views the activity of modeling as the central issue for both knowledge acquisition and machine learning, the integration of machine learning into knowledge acquisition follows naturally. By inspecting the properties demanded of systems which support and perform modeling we can conclude:

- Modeling is an interactive process, thus the system should be interactive at all points in the modeling process
- Models can be operational, thus the system should not only store the model but also answer questions about the knowledge
- Modeling is a scientific process, thus the system's behavior can be designed along the lines of theory formation: gathering data, laying out a representational framework, finding regularities (laws), predicting facts, verifying the predictions, and starting all over again.

The most important implication of our observations on modeling is the need for **reversibility** because of the tentative character of layout and even laws. Starting with a sloppy model and then improving with revisions it has been illustrated by examples from BLIP, a system designed to meet the modeling requirements.

Acknowledgements

I'd like to warmly thank Bernhard Nebel, with whom I discussed the possibilities of change in KL-ONE alike systems, Marianne LaFrance who frankly spoke about her experience in knowledge elicitation at last year's workshop in Banff Knowledge Acquisition for Knowledge-based Systems, Brigitte Bartsch-Spoerl, who encouraged me when I first thought about the sloppy modeling paradigm, and all participants of the workshop on Knowledge Representation and Organization in Machine Learning, who enriched my understanding of the modeling issue by lively discussions.

REFERENCES

Amarel, S.(68): "On Representation of Problems of Reasoning about Actions"; In: B. Meltzer, D. Michie (eds.): Machine Intelligence, Vol. 3, pp. 131-171, Edinburgh University Press, Edinburgh, 1968

Boose, J.H., Bradshaw, J.M. (87): "Expertise Transfer and Complex Problems Using AQUINAS as a Knowledge Acquisition Workbench for Expert Systems"; In: International Journal of Man-Machine Studies, Vol. 26, No. 1, pp. 3-28, January 1987

Brachman, R.J.; Schmolze, J.G. (85): "An Overview of the KL-ONE Knowledge Representation System"; In: Cognitive Science 9, 2, pp.171-216.

Buchanan, B.G.; Mitchell, T.M. (78): "Model-Directed Learning of Production Rules"; In: D.A. Waterman, F. Hayes-Roth (eds.): Pattern-Directed Inference Systems, pp. 297-312, Academic Press, New York, San Francisco, London 1978

Carbonell, J.G. (83): "Derivational Analogy and its Role in Problem Solving"; In: Proceedings of the AAAI, Washington, D.C. 1986, pp. 64-69

Clancey, W.J. (86): "From GUIDON to NEOMYCIN and HERACLES"; In: The AI Magazine, Vol. 2, No. 3, pp. 40-60

Cox, L.A.; Blumenthal, R. (87): "KRIMB: An Intelligent Knowledge Acquisition and Representation Program for Interactive Model Building"; In: Proceedings of the first European Workshop on Knowledge Acquisition for Knowledge-Based Systems, p.E3, Reading University, UK, September 1987

Dietterich, T.G. (86): "The EG Project: Recent Progress"; In: T.M. Mitchell, J.G. Carbonell, R.S. Michalski (eds.): Maschine Learning, a Guide to Current Research, pp. 51-54, Kluver Academic Press, Boston, Dordrecht, Lancaster 1986

Diederich, J.; Ruhmann, I.; May, M. (87): "KRITON: a knowledge-acquisition tool for expert systems"; In: International Journal of Man-Machine Studies, Vol. 26, No. 1, pp. 29-40, January 1987

Dietterich, T.G.; Michalski, R.S. (81): "Inductive Learning of Structural Descriptions: Evaluation Criteria and Comparative Review of Selected Methods"; In: Artificial Intelligence, Vol. 16, No. 3, pp. 257-294, July 1981

Dietterich, T.G.; Michalski, R.S. (85): "Discovering Patterns in Sequences of Events"; In: Artificial Intelligence, Vol. 25, No. 2, pp. 187-232, February 1985

Emde, W. (84): "Inkrementelles Lernen mit heuristisch generierten Modellen"; Technical University Berlin, Department of Applied Computer Science, KIT-Report No. 22

Emde, W. (87a): "Non-Cumulative Learning in METAXA.3"; In: Proceedings of the 10th IJCAI-87, Milano 1987, pp. 208-210

Emde, W. (87b): "An Inference Engine for Representing Multiple Theories"; (In this Volume)

Eshelman, L., Ehret, D., McDermott, J., Tan, M. (87): "MOLE: A Tenacious Knowledge Acquisition Tool"; In: International Journal of Man-Machine Studies, Vol. 26, No. 1, pp. 41-54, January 1987

Gaines, B.R. (87): "Knowledge Acquisition for Expert Systems"; In: Proceedings of the first European Workshop on Knowledge Acquisition for Knowledge-Based Systems, p.A3, Reading University, UK, September 1987

Garfinkel, H. (67): "Studies in Ethnomethodology"; Prentice-Hall, Englewood Cliffs, N.J.

Garfinkel, H. (73): "Das Alltagswissen über soziale und innerhalb sozialer Strukturen"; In: Alltagswissen, Interaktion und gesellschaftliche Wirklichkeit; rororo studium, Reinbeck bei Hamburg, pp. 189-262, 1973

Hayes-Roth, F.; Waterman, D.A.; Lenat, D. (83): "Building Expert Systems"; Addison-Wesley, Reading, Massachusetts, 1983

Herkner, W. (Hg.) (80): "Attribution - Psychologie der Kausalität"; Hans Huber Verlag, Bern, Stuttgart, Wien 1980

Johnson, N. (87): "Mediating representations in Knowledge Elicitation"; In: Proceedings of the first European Workshop on Knowledge Acquisition for Knowledge-Based Systems, p.A2, Reading University, UK, September 1987

Kelly, G.A. (55): "The Psychology of Personal Constructs"; Norton, New York, 1955

Kietz, J.-U. (88): "Incremental and Reversible Acquisition of Taxonomies"; in: Proceedings of the European Knowledge Acquisition Workshop (EKAW 88), p.24, GMD-Studien, Bonn, June 1988

Kodratoff, Y. (86): "Is AI a Sub-Field of Computer Science? Or AI is the Science of Explanations"; Universite de Paris-Sud, Laboratoire de Recherche en Informatique, Report No. 312

Kodratoff, Y.; Tecuci, G. (87a): "DISCIPLE-1: Interactive Apprentice System in Weak Theory Fields"; In: Proceedings of the 10th IJCAI-87, Milano 1987, pp. 271-273

Kodratoff, Y., Tecuci, G. (87b): "The Central Role of Explanations in DISCIPLE" (in this volume)

LaFrance, M. (87): "The Knowledge Acquisition Grid: a method for training knowledge engineers"; In: International Journal of Man-Machine Studies, Vol. 26, No. 2, pp. 245-256, February 1987

Laird, J; Rosenbloom, P.; Newell, A. (86): "Universal Subgoaling and Chunking"; Kluver Academic Publishers, Boston, Dordrecht, Lancaster, 1986

Langley, P.; Bradshaw, G.L.; Simon, H.A. (83): "Rediscovering Chemistry With the BACON System"; In: T.M. Mitchell, J.G. Carbonell, R.S. Michalski (eds.): Maschine Learning, an Artificielle Intelligence Approach; pp. 307-330, Tioga Publ., Palo Alto, CA, 1983

Michalski, R.S. (80): "Knowledge Acquisition through Conceptual Clustering: A Theoretical Framework and an Algorithm for Partitioning Data into Conjunctive Concepts"; In: University of Illinois, Urbana-Champaign, Dept of Computer Science Report No. 80-1026 (1980)

Michalski, R.S., Chilausky, R.L.(80): "Learning by Being Told and Learning from Examples: An Experimental Comparison of the Two Methods of Knowledge Acquisition in the Context of Developing an Expert System for Soybean Disease Diagnosis"; International Journal of Policy Analysis and Information Systems, Vol. 4, No. 2, pp. 125-161, 1980

Michalski, R.S./Stepp, R.E.(83): "Learning from Observation: Conceptual Clustering" In: T.M. Mitchell, J.G. Carbonell, R.S. Michalski (eds.): Maschine Learning, an Artificielle Intelligence Approach; pp. 331-364, Tioga Publ., Palo Alto, CA, 1983

Michie, D. (82): "Game-playing Programs and the Conceptual Interface" In: SIGART Newsletter, No. 80, pp. 64-70, 1982

Michie, D.; Bratko, I. (78): "Advice Table Representations of Chess End-game Knowledge"; In: Proceedings of the AISB/GI Conference on Artificial Intelligence, Hamburg 1978

Mitchell, T.M. (82): "Generalisation as Search"; In: Artificial Intelligence, Vol. 18, No. 2, pp. 203-226, March 1982

Mitchell, T.M.; Mahadevan, S.; Steinberg, L.I. (85): "LEAP: A Learning Apprenticeship for VLSI Design"; In: Proceedings of the 9th IJCAI-85, Los Angeles 1985, pp. 573-580

Morik, K.J. (87): "Acquiring Domain Models"; In: International Journal of Man-Machine Studies, Vol. 26, No. 1, pp. 93-104, January 1987

Mostow, J.; Swartout, B. (86): "Towards Explicit Integration of Knowledge in Expert Systems: An Analysis of MYCIN's Therapy Selection Algorithm"; In: Proceedings of the AAAI-86, Philadelphia 1986, pp. 928-935

Musen, M.A.; Fagan, L.M.; Combs, D.M.; Shortliffe, E.H. (87): "Use of a domain model to drive an interactive knowledge-editing tool"; In: International Journal of Man-Machine Studies, Vol. 26, No. 1, pp. 105-121, January 1987

Nebel, B., von Luck, K. (87): "Issues of Integration and Balancing in Hybrid Knowledge Representation Systems"; In: Morik (ed.): Proceedings of the GWAI-87, 11thGerman Workshop on Artificial Intelligence, Guericke; pp. 114 - 123, Springer Verlag, Berlin, 1987

Neches, R./Swartout, W.R./Moore, J.(85): "Enhanced Maintenance and Explanation of Expert Systems through Explicit Models of their Development"; In: IEEE Transactions on Software Engineering, Vol. SE-11, No. 11 November, pp. 1337 - 1351, 1985

Pazzani, M.J. (86): "Explanation-Based Learning for Knowledge based Systems"; In: International Journal of Man-Machine Studies, Vol. 26, No. 4, pp. 453-472, January 1987

Pazzani, M.J. (87): "Creating High Level Knowledge Structure from Primitive Elements"; (in this volume)

Quinlan, R.J. (83): "Learning Efficient Classification Procedures and their Application to Chess End Games"; In: T.M. Mitchell, J.G. Carbonell, R.S. Michalski (eds.): Maschine Learning, an Artificielle Intelligence Approach; pp. 463-482, Tioga Publ., Palo Alto, CA, 1983

Quinlan, R.J. (86): "Induction of Decision Trees"; In: Maschine Learning, Vol. 1, No. 1, pp. 81-106, 1986

Rajamoney, S.; DeJong, G.; Faltings, B. (85): "Towards a Model of Conceptual Knowledge Acquisition through Directed Experimentation"; In: Proceedings of the 9th IJCAI-85, Los Angeles 1985, pp. 688-690

Rose, D.; Langley, P. (86): "STAHLp: Belief Revision in Scientific Discovery"; In: Proceedings of the AAAI 86, Philadelphia 1986, pp. 528-532

Schank, R.C. (82): "Looking at Learning"; In: Proceedings of the ECAI-82, Orsay Paris 1982, pp. 11 - 18

Schütz, A. (62): "Collected papers"; Nijhoff, Den Haag, 1962

Shortliffe, E.H. (76): "Computer-Based Medical Consultations: MYCIN"; American Elsevier Publishing Company, 1976

Swartout, W.R. (83): "XPLAIN: A System for Creating and Explaining Expert Consulting Programs"; In: Artificial Intelligence, Vol. 21, No. 3, pp. 285-325, September 1983

Thieme, S. (87): "The Acquisition of Model-Knowledge for a Model-Driven Machine Learning Approach"; (In this Volume)

Turkle, S. (84): "The Second Self. Computers and the Human Spirit"; Simon and Schuster, New York, 1984

Wilkens, D.C. (86): "Knowledge Base Debugging Using Apprenticeship Learning Techniques"; In: J. Boose, B. Gaines (eds.): Proceedings of the Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, 1986

Winston, P.H. (75): "Learning Structural Descriptions from Examples"; In: P.H. Winston (ed.): The Psychology of Computer Vision; McGraw-Hill, New York, 1975

Wisniewski, E.; Winston, H.; Smith, R.; Kleyn, M. (86): "Case Generation for Rule Synthesis"; In: J. Boose, B. Gaines (eds.): Proceedings of the Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, 1986

Wrobel, S. (87a): "Design Goals for Sloppy Modeling Systems"; In: J. Boose, B. Gaines (eds.): Proceedings of the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, 1987, (to appear in Int. Journal of Man-Machine Studies)

Wrobel, S. (87b): "Higher-order Concepts in a Tractable Knowledge Representation"; In: Morik (ed.): Proceedings of the GWAI-87, 11thGerman Workshop on Artificial Intelligence, Guericke; pp. 129 - 138, Springer Verlag, Berlin, 1987

Wrobel, S. (87c): "Demand Driven Concept Formation"; (In this Volume)

Young, R.M.; Gammack, J. (87): "Role of Psychological Techniques and Intermediate Representations in Knowledge Elicitation"; In: Proceedings of the first European Workshop on Knowledge Acquisition for Knowledge-Based Systems, p.D7, Reading University, UK, September 1987

THE CENTRAL ROLE OF EXPLANATIONS IN DISCIPLE

Yves Kodratoff
U.A. 410 du C.N.R.S., Laboratoire de Recherche en Informatique
Bat.490, Université de Paris-Sud , 91405 Orsay Cedex, FRANCE
Gheorghe Tecuci
Research Institute for Computers and Informatics
71316, Bd. Miciurin 8-10, Sector 1, Bucharest ROMANIA

Abstract

DISCIPLE is a Knowledge Acquisition system that contains several learning mechanisms as recognized by Machine Learning. The central mechanism in DICIPLE is the one of **explanations** which is used in all the learning modes of DISCIPLE.
When using the *Explanation-Based* mode of learning, an explanation points at the most relevant features of the examples.
When using the *Analogy-Based* mode of learning, the explanations are used to generate instances analogous to those provided by the user.
When using the *Similarity-Based* mode of learning, the explanations are "examples" among which similarities are looked for.
The final result of DISCIPLE is the description of the validity domain of the variables contained in the rules. Since the users always provides totally instantiated rules, the system must automatically variabilize them, and then must find the validity domain of these variables by asking "clever" questions to the user. Given a particular (instantiated) rule by its user, the system will look in its Knowledge Base for possible explanations of this rule, and ask the user to validate them. The set of explanations validated by the user is then used as a set of (almost) sufficient conditions for the application of the instantiated rule.

1. Introduction

Knowledge Acquisition tools are expected to perform a wide range of tasks in order to help the human expert to give a computer-usable form to his knowledge. These tasks include decomposing problems, combining uncertain information, testing, help in defining data types, help in the expansion, refinement, and deficiencies recovery of the knowledge base, use of multiple sources of knowledge, changes in knowledge representation, development of models of the expert knowledge (see the series of