90-15

19

APPRENTICESHIP LEARNING IN IMPERFECT DOMAIN THEORIES

Gheorghe Tecuci
(Research Institute for Computers and Informatics, Bucharest))

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

Abstract

This chapter presents DISCIPLE, a multistrategy, integrated learning system illustrating a theory and a methodology for learning expert knowledge in the context of an imperfect domain theory. DISCIPLE integrates a learning system and an empty expert system, both using the same knowledge base. It is initially provided with an imperfect (nonhomogeneous) domain theory and learns problem-solving rules from the problem-solving steps received from its expert user, during interactive problem-solving sessions. In this way, DISCIPLE evolves from a helpful assistant in problem solving to a genuine expert. The problem-solving method of DISCIPLE combines problem reduction, problem solving by constraints, and problem solving by analogy. The learning method of DISCIPLE depends on its knowledge about the problem-solving step (the example) from which it learns. In the context of a complete theory about the example, DISCIPLE uses explanation-based learning to improve its performance. In the context of a weak theory about the example, it synergistically combines explanation-based learning, learning by analogy, empirical learning, and learning by questioning the user, developing its competence. In the context of an incomplete theory about the example, DISCIPLE learns by combining the above-mentioned methods, improving both its competence and performance.

19.1 INTRODUCTION

The present success of AI is mostly due to the knowledge-based systems that proved to be useful almost anywhere. As the name suggests, the power of a knowledge-based system comes from its knowledge. However, building a knowledge base for such a system is a very complex, time-consuming, and error-prone process. Moreover, the resulting system lacks or has only poor abilities to update its knowledge or to acquire new knowledge.

One promising solution to this "knowledge-acquisition bottleneck" is represented by the Learning Apprentice Systems (LAS). An LAS is an interactive knowledge-based consultant that is provided with an initial domain theory and is able to assimilate new problem-solving knowledge by observing and analyzing the problem-solving steps contributed by its users, through their normal use of the system [Mitchell, Mahadevan, and Steinberg, 1985].

Representative examples of this approach are the systems LEAP [Mitchell, Mahadevan, and Steinberg, 1985] and GENESIS [DeJong and Mooney, 1986]. LEAP's domain of expertise is the VLSI design and GENESIS's is story understanding. A common feature of LEAP and GENESIS is that they are based on a strong (complete) domain theory that allows them to learn a general rule or schemata from a single example by reducing learning to deductive reasoning.

Nevertheless, such beautifully tailored domains are seldom available. *A typical real-world domain theory is nonhomogeneous* in that it provides complete descriptions of some parts of the domain, and only incomplete or even poor (weak) descriptions of other parts of the domain. A learning episode, however, uses only one part of the domain theory; and this part may have the features of a complete, incomplete or weak theory even if, globally, the theory is nonhomogeneous. Therefore, a learning system should be able to learn a general rule or concept not only when disposing of a complete theory about an example, but also when disposing of an incomplete or even weak theory about it. An illustration of such a learning system is DISCIPLE. *DISCIPLE is a multistrategy, integrated learning system.* It has the same general purpose as a learning apprentice system, but it is based on a multistrategy approach to learning, instead of on deductive reasoning.

DISCIPLE is a tool for building practical expert systems. It integrates an empty expert system and a learning system, both using the same knowledge base. To build an expert system with DISCIPLE, one has to first introduce elementary knowledge about an application domain into DISCIPLE's knowledge base—knowledge constituting a nonhomogeneous theory of the domain. Next, DISCIPLE may be used to solve problems interactively, according to the following scenario:

The user gives DISCIPLE the problem to solve, and the expert subsystem starts solving this problem by showing the user each problem-solving step (which we shall

call *partial solution*). The user may agree with or reject it. In the latter case, or when DISCIPLE is unable to propose any partial solution, the user is compelled to give his own solution. Once this solution is given, a learning process will take place. DISCIPLE will try to learn a general rule so that, when faced with problems similar to the current one (which it has been unable to solve), it will become able to propose a solution similar to the solution, given by the user, to the current problem. In this way, DISCIPLE progressively evolves from a useful assistant in problem solving to a genuine expert.

19.2 DISCIPLE AS AN EXPERT SYSTEM

In DISCIPLE we have adopted a *problem-reduction* approach to problem solving. That is, a problem is solved by successively reducing it to simpler subproblems. This process continues until the initial problem is reduced to a set of elementary problems; that is, problems with known solutions. Moreover, the problem to solve may be initially imprecisely formulated, becoming better and better formulated as the problem-solving process advances. To this purpose, DISCIPLE formulates, propagates, and evaluates constraints [Tecuci, 1988; Tecuci, *et al.*, 1987].

Problem reduction is a general method, suitable for solving a large variety of problems. In the following, however, we shall consider only problems of designing action plans for achieving partially specified goals. These problems are similar to those solved by PLANX10 [Sridharan and Bresina, 1982], NONLIN [Tate, 1977], and others. An example of such a problem is:

- given the incomplete specifications of a loudspeaker;
- design the actions needed to manufacture the loudspeaker.

DISCIPLE may start with the following top-level operation, which can be seen as the current goal:

MANUFACTURE OBJECT loudspeaker

It will try to solve this problem by successive decompositions and specializations, as illustrated in Figure 19–1 and in Figure 19–2. DISCIPLE will combine such decompositions and specializations, building a problem-solving tree like the one in Figure 19–3. This process continues until all the leaves of the tree are elementary actions, that is, actions that can be executed by the entity manufacturing the loudspeaker.

Figure 19–3 shows a standard AND tree, the solution to the problem from the top of this tree consisting of the leaves of the tree. That is, to manufacture the loudspeaker, one has to perform the following sequence of operations:

FIX OBJECTS contacts ON chassis

In order to solve the problem
MANUFACTURE OBJECT loudspeaker
solve the subproblem

1. MAKE OBJECT chassis-assembly
   In order to solve this subproblem solve the sub-subproblems
   1.1 FIX OBJECT contacts ON chassis
   1.2 MAKE OBJECT mechanical-chassis-assembly
   1.3 FINISHING-OPERATIONS ON entrefer
      In order to solve this subproblem solve the sub-subproblems
      1.3.1 CLEAN OBJECT entrefer
      1.3.2 VERIFY OBJECT entrefer
2. MAKE OBJECT membrane-assembly
3. ASSEMBLE OBJECT chassis-assembly WITH membrane-assembly
   In order to solve this subproblem solve the sub-subproblems
   3.1 ATTACH OBJECT membrane-assembly ON chassis-assembly
   3.2 ATTACH OBJECT ring ON chassis-membrane-assembly
      In order to solve this subproblem solve the sub-subproblems
      3.2.1 APPLY OBJECT mowicoll ON ring
      3.2.2 PRESS OBJECT ring ON chassis-membrane-assembly
4. FINISHING-OPERATIONS ON loudspeaker

Figure 19–1: Problem-solving operations: Decompositions of problems into simpler subproblems

In order to solve the problem
CLEAN OBJECT entrefer
solve the specialization
CLEAN OBJECT entrefer WITH air-jet-device
In order to solve this problem solve the specialization
CLEAN OBJECT entrefer WITH air sucker
In order to solve the problem
APPLY OBJECT mowicoll ON ring
solve the specialization
APPLY OBJECT mowicoll-C107 ON ring

Figure 19–2: Problem-solving operations: specializations of problems

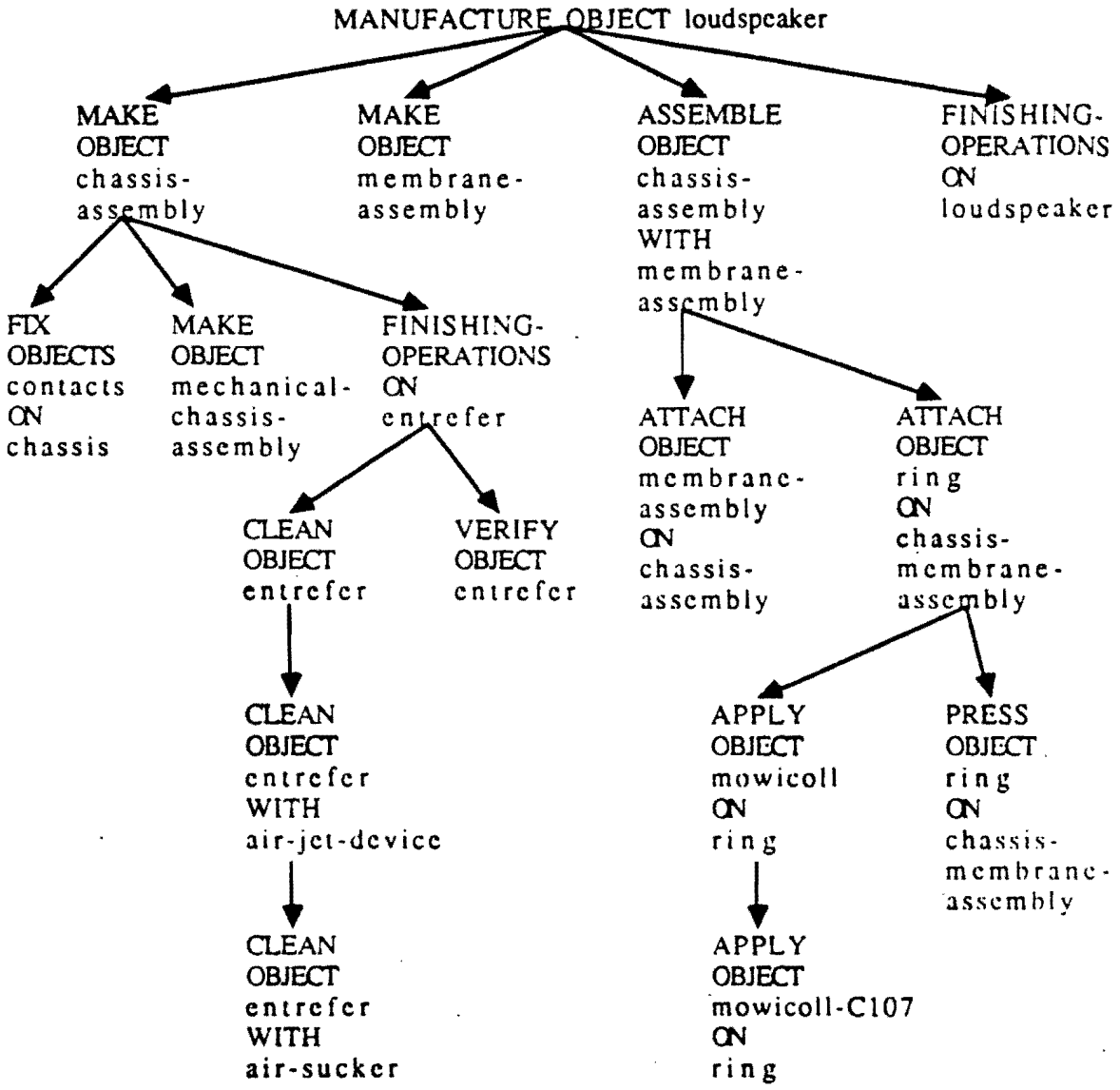


Figure 19–3: A problem-solving tree. It was built by using the decompositions and the specializations from Figures 19–1 and 19–2.

MAKE OBJECT mechanical-chassis-assembly
CLEAN OBJECT entrefer WITH air-sucker
VERIFY OBJECT entrefer
MAKE OBJECT membrane-assembly
ATTACH OBJECT membrane-assembly ON chassis-assembly
APPLY OBJECT mowicoll-C107 ON ring
PRESS OBJECT ring ON chassis-membrane-assembly
FINISHING-OPERATIONS ON loudspeaker

Let us notice that the decompositions and the specializations model, in fact, the main operations used in design, where one usually starts with a very general specification of an object, successively imposes different constraints on the specification, and reduces object design to subparts design.

19.3 THE LEARNING PROBLEM

The decompositions and the specializations from Figure 19–3 were the result of the application of general reduction rules or were directly indicated by the user. From each solution received from the user, DISCIPLE is trying to learn a general problem-solving rule. Therefore, the learning problem of DISCIPLE may be formulated as shown in Figure 19–4.

For instance,

Given:
The theory of loudspeaker manufacturing;
The problem of attaching two parts of the loudspeaker (the 'ring' and the 'chassis-membrane-assembly') *and the decomposition of this problem into two simpler subproblems* expressing the gluing of the two parts with 'mowicoll' (see Figure 19–5).

Determine:
A general decomposition rule indicating the conditions under which one may reduce an 'attachment' problem to a process of gluing (see Figure 19–6).

As one may notice, the structure of **General Rule 1** in Figure 19–6 is identical with the structure of **Example 1** in Figure 19–5. Therefore, rule learning is reduced to learning the features that the objects 'x', 'y', and 'z' should have so that the attachment of 'x' and 'y' may be reduced to a process of gluing them with 'z'. Otherwise stated, one should learn the concepts represented by these objects.

The method of learning this rule depends on the system's theory (knowledge) about **Example 1**. We distinguish between three types of theories: *complete*, *weak*, and *incomplete*.

A *complete theory* about Example 1 consists of the complete descriptions of the objects and actions from this problem-solving episode. In such a case, DISCIPLE uses an explanation-based learning method, being able to learn at once a general rule from Example 1 alone.

A *weak theory* about Example 1 consists only of incomplete descriptions of the objects. It differs qualitatively from a complete theory in that it does not contain action descriptions. In this case, DISCIPLE uses an interactive learning method that synergistically combines explanation-based learning, learning by analogy, empirical learning, and learning by questioning the user.

Given:

Domain Theory

The domain theory contains:

- a specification of the types of objects in the world and their properties and relations;
- a set of inference rules for inferring properties and relations from other properties and relations;
- a set of action models describing the actions that may be performed in the domain. An action model specifies the preconditions of the action (i.e., the states of the world in which the action may be executed), the effects of the action (i.e., the states that result after the execution of the action), as well as the objects that may play certain roles in the action (the agent executing the action, the object on which the action is performed, the instrument used, etc.).

Problem-Solving Episode

It consists of

- P, a problem to solve, and
- S, a (partial) solution to P.

Determine:

A General Problem-Solving Rule.

According to this rule, problems similar to P will receive solutions similar to S.

Figure 19–4: The learning problem of DISCIPLE

Example 1:

Solve the problem

ATTACH OBJECT ring ON chassis-membrane-assembly

by solving the subproblems

APPLY OBJECT mowicoll ON ring

PRESS OBJECT ring ON chassis-membrane-assembly

Figure 19–5: A decomposition indicated by the user

The intermediate case, between a complete theory and a weak theory, is the *incomplete theory*. It contains incomplete descriptions of the objects and the actions from Example 1. In the case of an incomplete theory about Example 1, DISCIPLE learns a general rule by combining the method corresponding to the weak theory with the one corresponding to the complete theory.

IF

(x TYPE solid) & (y TYPE solid) & (x PARTIALLY-FITS y) &
(z ISA adhesive) & (z TYPE fluid) & (z GLUES x) & (z GLUES y)

THEN

General Rule 1:

solve the problem

ATTACH OBJECT x ON y

by solving the subproblems

APPLY OBJECT z ON x
PRESS OBJECT x ON y

Figure 19–6: The general decomposition rule learned from Example 1: If 'x' and 'y' are two solid objects that partially fit each other, and there is a fluid adhesive 'z' that glues both 'x' and 'y', then one may attach 'x' on 'y' by first applying 'z' on 'x' and then by pressing 'x' on 'y'

A side effect of rule learning in the context of a weak or incomplete theory is that of developing the domain theory. In the following sections we shall present these three learning methods of DISCIPLE.

19.4 LEARNING IN A COMPLETE THEORY DOMAIN

19.4.1 A Sample of a Complete Theory

In the case of DISCIPLE, a complete theory of a domain consists of complete descriptions of all the objects and actions of the domain. In particular, a complete theory about the problem-solving episode in Figure 19–5, contains the complete descriptions of the objects 'ring', 'chassis-membrane-assembly', and 'mowicoll', as well as the complete descriptions (models) of the actions 'ATTACH', 'APPLY', and 'PRESS'.

The objects are described by specifying all the relevant factual properties and relations. Some of these may be explicitly specified, as indicated in Figure 19–7.

Other properties and relations may be implicitly specified by using inference rules for deducing them from other properties and relations, as indicated in Figure 19–8.

The action models describe the actions that may be performed in the domain. A complete action model specifies all the necessary preconditions of the action (i.e., all the states of the world in which the action may be executed), all its effects (i.e., the states that result after the execution of the action), as well as all the objects that may play certain roles in the action (the agent executing the action, the object on which the action is performed, the instrument used, etc.).

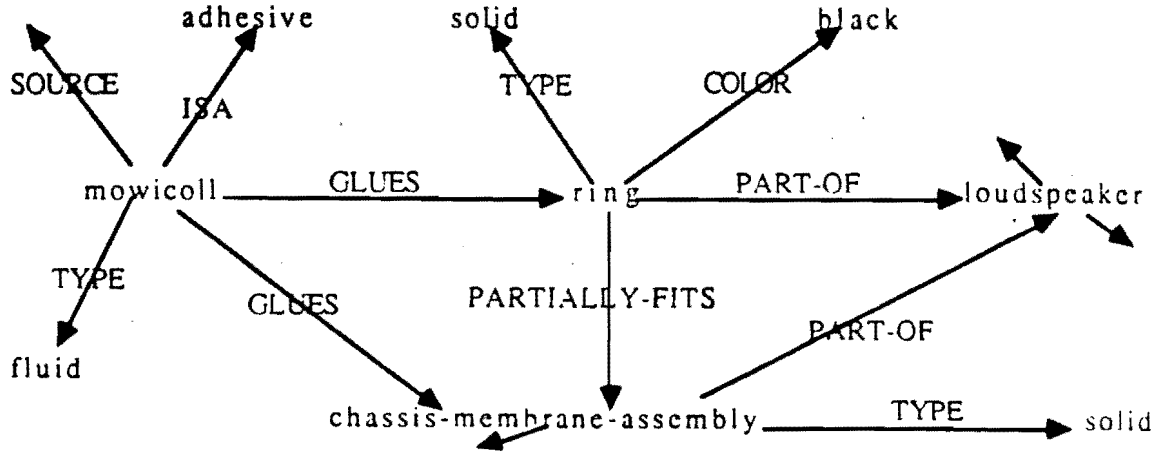


Figure 19–7: A hierarchical semantic network containing explicit representations of object properties and relations

$$\forall x\ \forall y\ [(x \text{ GLUED-ON } y) \Rightarrow (x \text{ ATTACHED-ON } y)]$$
$$\forall x\ \forall y\ \forall z\ [(z \text{ ISA adhesive}) \& (z \text{ GLUES } x) \& (z \text{ GLUES } y) \& (z \text{ BETWEEN } x\ y) \Rightarrow (x \text{ GLUED-ON } y)]$$
$$\forall x\ \forall y\ [(x \text{ GLUES } y) \Rightarrow (x \text{ ADHERENT-ON } y)]$$

Figure 19–8: Inference rules for deducing new properties and relations of objects

Action	Preconditions	Effects
ATTACH OBJECT x ON y	(x TYPE solid) & (y TYPE solid)	(x ATTACHED-ON y)
APPLY OBJECT z ON x	(z TYPE fluid) & (z ADHERENT-ON x) & (x TYPE solid)	(z APPLIED-ON x)
PRESS OBJECT x ON y	(z APPLIED-ON x) & (x PARTIALLY-FITS y) & (y TYPE solid)	(z BETWEEN x y)

Figure 19–9: Action models

Figure 19–9 presents the models of the actions from the problem-solving episode in Figure 19–5. For instance, the action 'APPLY' may be performed if and only if 'x' is a solid object and 'z' is a fluid object that is adherent on 'x'. As an effect of performing this action, 'z' will be applied on 'x'. Notice that the necessary features of the objects are specified in the action's preconditions.

19.4.2 General Presentation of the Learning Method

In the case of a complete theory about Example 1, the learning method of DISCIPLE follows the explanation-based learning paradigm developed by [DeJong and Mooney, 1986; Fikes, Hart, and Nilsson, 1972; Mitchell, Keller, and Kedar-Cabelli, 1986] and others:

1. Prove that the solution indicated by the user is indeed a solution of the problem to solve. This proof isolates the relevant features of the objects in Example 1; that is, those features that will be present in the condition of General Rule 1.
2. Generalize the proof tree as much as possible so that the proof still holds. This is done as in [Mooney and Bennet, 1986] by replacing each instance of action model or inference rule with its general pattern and by unifying these patterns. By generalizing the proof tree, one generalizes the problem, its solution, and the relevant features.
3. Formulate the learned rule from the generalized proof by extracting the generalized problem, its generalized solution, and the generalized relevant features, which constitute the applicability condition of the rule.

In the following sections we shall briefly illustrate this method with the aid of **Example 1** (Figure 19–5).

19.4.3 Proving the Example

To prove **Example 1** means to show that the sequence of the actions

APPLY OBJECT mowicoll ON ring
PRESS OBJECT ring ON chassis-membrane-assembly achieves
the goal of the action
ATTACH OBJECT ring ON chassis-membrane-assembly that is,
achieves the goal
(ring ATTACHED-ON chassis-membrane-assembly).

The proof is indicated in Figure 19–10. It was obtained by using the object descriptions in Figure 19–7, the inference rules in Figure 19–8, and the action models in Figure 19–9.

The leaves of the tree in Figure 19–10 are those features of 'ring', 'chassis-membrane-assembly', and 'mowicoll' that allowed one to reduce the problem of attaching the 'ring' on the 'chassis-membrane-assembly' to the process of gluing them with 'mowicoll'. Thus, by proving the example, one isolates the relevant features of it (see Figure 19–11). The 'color' of the 'ring' or the 'source' of the 'mowicoll' were not useful in proving the validity of the example. Therefore, these features are not important for this example.

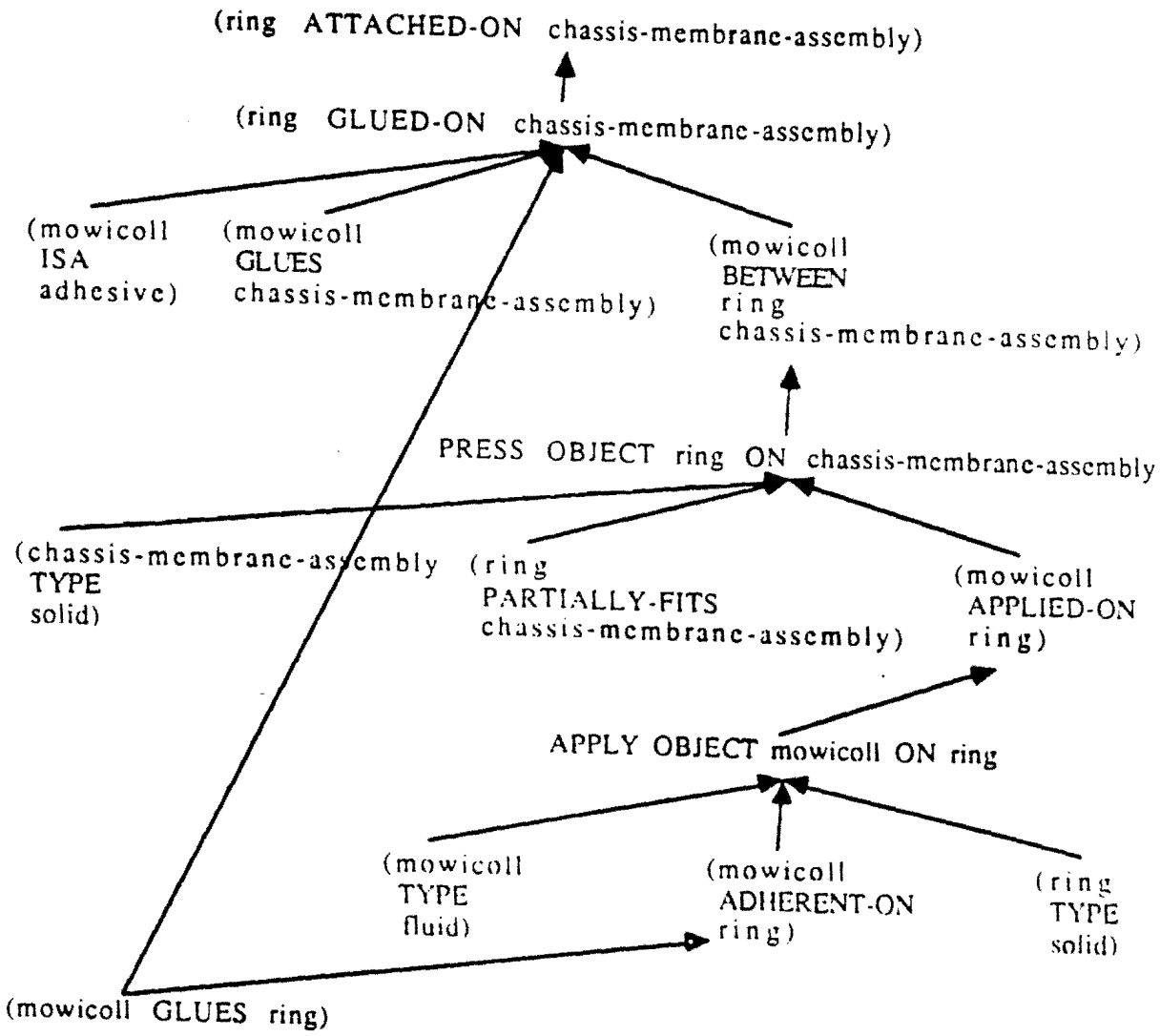


Figure 19–10: A complete proof of Example 1

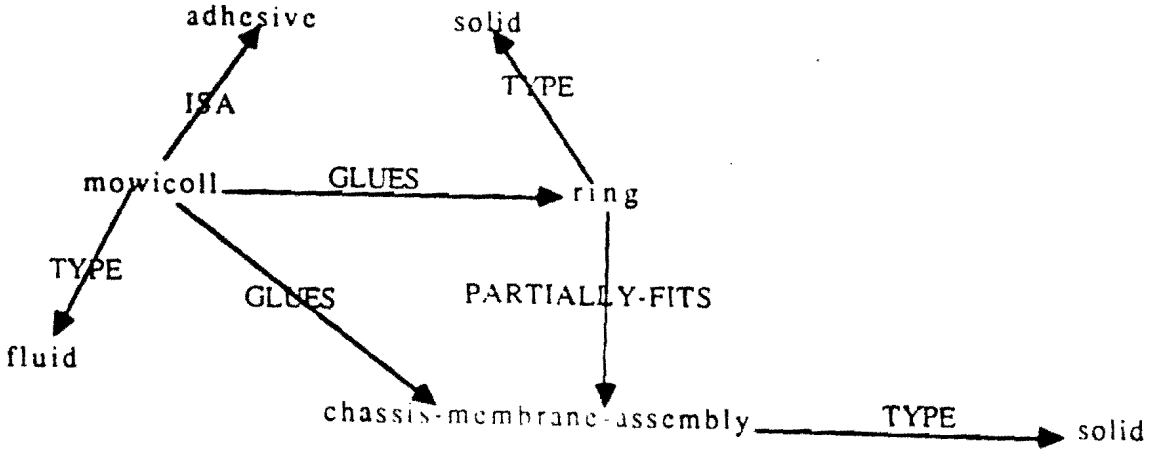


Figure 19–11: The relevant features of Example 1

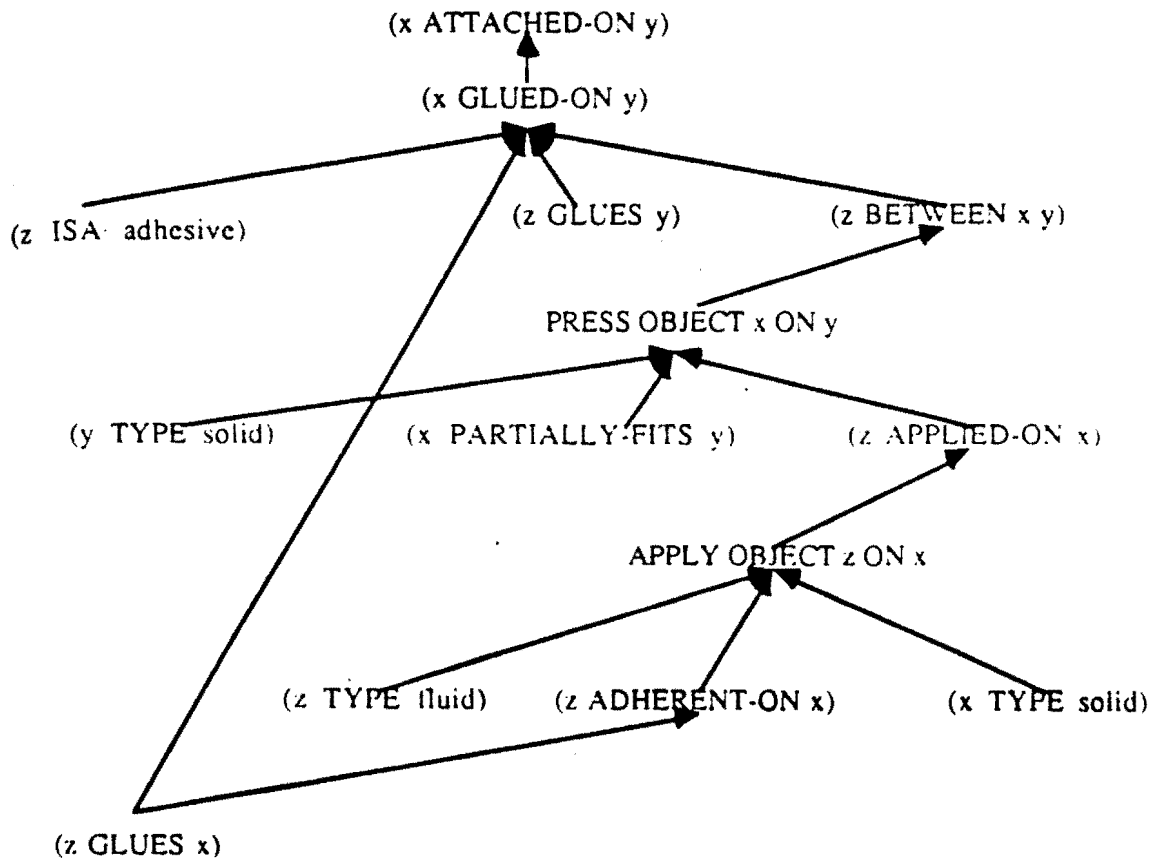


Figure 19–12: The generalization of the proof in Figure 19–10

19.4.4 Generalization of the Proof

The next step consists in the generalization of the proof, as much as possible, so that the proof still holds. Since the proof in Figure 19–10 was obtained by using instances of inference rules and action models, one may generalize the proof by generalizing these instances. One way to do this is to first replace each instantiated inference rule or action model with its general pattern, and then to unify these patterns [Mooney and Bennet, 1986] (see Figure 19–12). The leaves of this generalized tree represent a justified generalization of the relevant features in Figure 19–11:

(x TYPE solid) & (y TYPE solid) & (x PARTIALLY-FITS y) &
(z ISA adhesive) & (z TYPE fluid) & (z GLUES x) & (z GLUES y)

They also represent a general precondition for which the sequence of the actions 'APPLY OBJECT z ON x', 'PRESS OBJECT x ON y' achieves the goal of the action 'ATTACH OBJECT x ON y'. That is, one has learned the general decomposition rule in Figure 19–6.

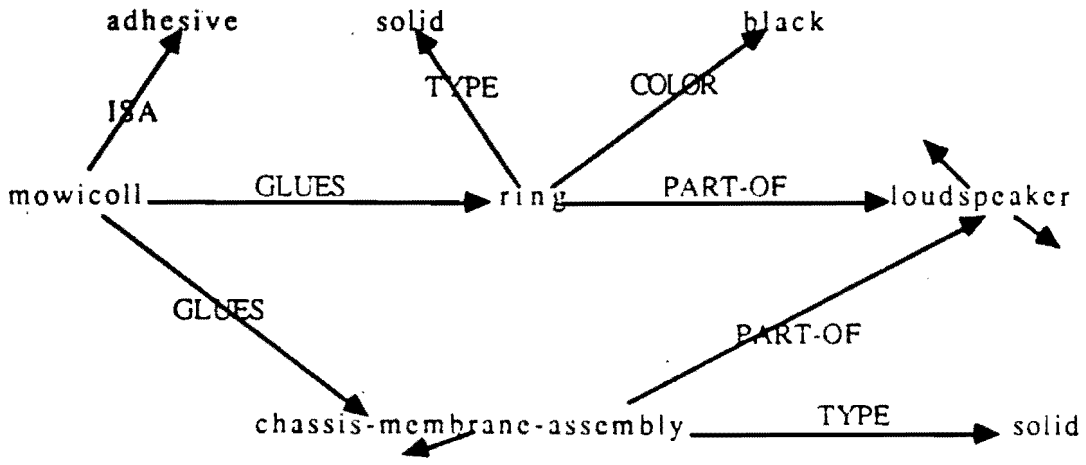


Figure 19–13: Fragment of a weak theory

19.5 LEARNING IN A WEAK THEORY DOMAIN

19.5.1 A Sample of a Weak Theory

A weak theory about the problem-solving episode in Figure 19–5 (Example 1) consists of the incomplete descriptions of the objects from this episode. It does not contain any action model. A sample of such a theory is represented in Figure 19–13.

Considering such a theory is justified because it is very difficult for an expert to describe the actions in terms of their preconditions and effects. On the other hand, it is much easier for him to describe the objects and to give examples of decompositions and specializations.

Therefore, instead of forcing the expert to completely formalize his knowledge, we decided to accept the theory that was easily provided by him and to learn the rest of the necessary knowledge.

19.5.2 General Presentation of the Learning Method

In the context of a weak theory, DISCIPLE will try to balance the lack of knowledge by using an integrated learning method whose power comes from the synergism of different learning paradigms: explanation-based learning, learning by analogy, empirical learning, and learning by questioning the user. Rule learning takes place in several stages, which are illustrated in Figure 19–14.

More formally, the learning method is the following one:

Explanation-based Mode

1. Find an explanation of the user's solution (Example 1) and call it Explanation 1.

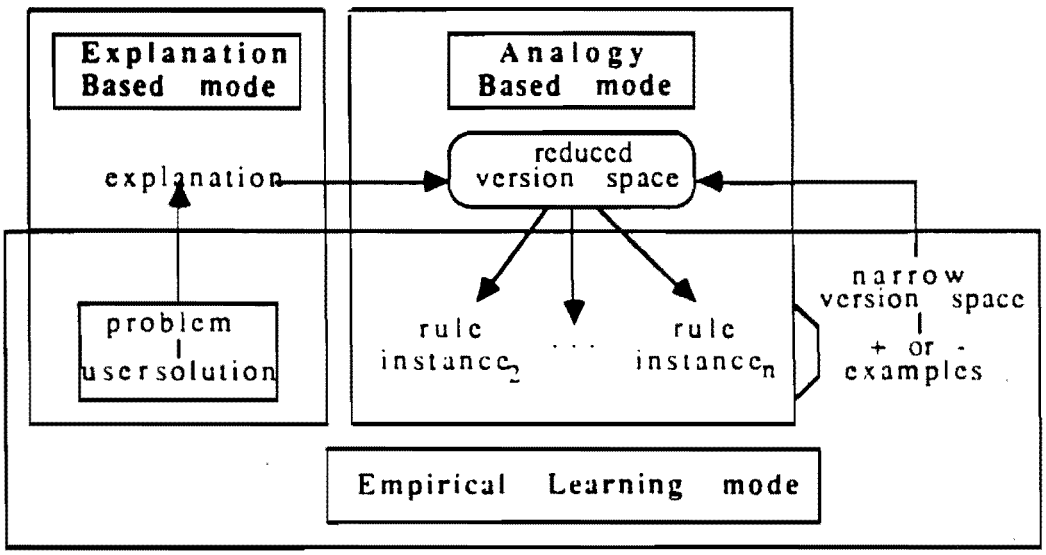


Figure 19–14: The learning method in the context of a weak theory. First DISCIPLE looks for a shallow explanation of the user's solution. Then it uses this explanation to formulate a reduced version space for the rule to be learned. Each rule in this space covers only instances analogous to the user's example. DISCIPLE carefuly generates analogous instances to be characterized as positive examples or as negative examples by the user. These are used to further narrow the version space until it contains only the rule illustrated by the user's solution.

Analogy-based Mode

2. Overgeneralize Example 1 by simply turning all the objects into variables, and call it General Rule 1.
3. Take Explanation 1 as a Lower Bound for the applicability condition of General Rule 1.
4. Overgeneralize Explanation 1 to the most general expression that may still be accepted by the user as an explanation of General Rule 1.
5. Take the overgeneralized explanation as an Upper Bound for the applicability condition of General Rule 1. The Upper Bound, the Lower Bound, and the General Rule 1 define a reduced version space for the rule to be learned.
6. Look in the knowledge base for tuples of objects that satisfy the Upper Bound but do not satisfy the Lower Bound.

   If there are such objects then call Explanation-i the properties of these objects that were used to prove that they satisfy the Upper Bound and go to step 7.

   If there are no such objects then show the Upper Bound, the Lower Bound, and the General Rule 1 to the user as an uncertain rule and stop.

7. Use the objects found in step 6 to generate an instance of General Rule 1. Call it Instance-i. This instance is analogous to Example 1.
8. Propose Instance-i to the user and ask him to characterize it as a valid or as an invalid reduction. If Instance-i is rejected by the user then go to step 9. Otherwise go to step 14.

Explanation-based Mode

9. Take Instance-i as a near miss (negative example) of the rule to be learned.
10. Find an explanation of why Instance-i was rejected by the user and call it Failure-Explanation-i.

Empirical Learning Mode

11. Specialize the Upper Bound as little as possible, so that not to cover Failure-Explanation-i.

    If the new Upper Bound is identical with the Lower Bound then take it as a necessary and sufficient condition of General Rule 1, show them to the user and stop, else go to step 12.
12. Specialize (if necessary) the Lower Bound as little as possible, so that not to cover Failure-Explanation-i.
13. Go to step 6.
14. Take Instance-i as a new positive example of the rule to be learned and Explanation-i as a true explanation of Instance-i.
15. Look for a maximally specific common generalization of the Lower Bound and Explanation-i. Two cases may occur:
    - if this generalization is not identical with the Upper Bound, then take it as the new Lower Bound and go to step 6;
    - if this generalization is identical with the Upper Bound, then take it as a necessary and sufficient condition of General Rule 1, show them to the user and stop.

In the following sections we shall illustrate and justify this learning method by using again Example 1 from Figure 19–5.

19.5.3 Explanation-based Mode

In its first learning step, DISCIPLE enters the explanation-based mode and tries to find an explanation (within its weak domain theory) of the validity of the solution in Figure 19–5.

We shall first define what we mean by an explanation in a weak theory and then we shall indicate a heuristic method to find such explanations.

19.5.3.1 Explanations in a Weak Theory Domain

Let 'P' be the problem to solve and 'S' a solution to this problem. As has been shown in Section 19.4, an explanation of the problem-solving episode 'solve P by S' is a proof that 'S' solves 'P'.

In the case of a complete theory about this problem-solving episode, the learning system is able to find itself such a proof. In the case of a weak theory, however, the system is no longer able to find such a proof because it lacks the models of the actions from 'P' and 'S'. In such a case, the explanation may be regarded as being the premise of a single inference whose conclusion is 'S solves P'.

For instance, in the context of a weak theory, a complete explanation of the problem-solving episode in Figure 19–5 would be the network from Figure 19–11. Indeed, the fact that the 'ring', the 'chassis-membrane-assembly', and the 'mowicoll' have the features in Figure 19–11 "explains" (in a weak theory) why the process of gluing the 'ring' and the 'chassis-membrane-assembly' with 'mowicoll' solves the problem of attaching them together.

19.5.3.2 A Heuristic to Find Explanations

The explanation of Example 1 consists of the leaves of the proof tree in Figure 19–11. Since such a tree cannot be built in a weak theory, DISCIPLE uses heuristics to propose plausible partial explanations to be validated by the user who may herself indicate other pieces of explanations. One heuristic is to *look for an explanation expressible in terms of the relations between the objects from the example, ignoring object features*. Therefore, to find an explanation of Example 1, DISCIPLE will look in its knowledge base for the links and for the paths (i.e., sequences of links) connecting 'ring', 'chassis-membrane-assembly', and 'mowicoll', and will propose the found connections as pieces of explanations of the Example 1. It is the user's task to validate them as true explanations:

Do the following justify your solution:
mowicoll GLUES ring? *Yes*
mowicoll GLUES chassis-membrane-assembly? *Yes*
ring PART-OF loudspeaker &
chassis-membrane-assembly PART-OF loudspeaker? *No*

All the pieces of explanations marked by a user's yes form the explanation of Example 1 (see Figure 19–15).

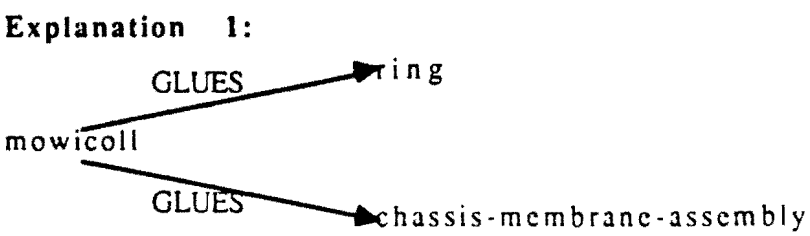


Figure 19–15: The explanation of Example 1

Notice that this explanation is incomplete. This is partially a consequence of using heuristics, and partially a consequence of the incompleteness of the domain theory (which may not contain all the relevant object properties and relations). Nevertheless, it shows some important features of the objects, features justifying the user's solution.

This explanation will be used in the next learning mode (the analogy-based mode), which will be described in the following section. There we shall also give a justification of the heuristic presented above.

19.5.4 Analogy-based Mode

The central intuition supporting the learning by analogy paradigm is that if two entities are similar in some respects then they could be similar in other respects as well. An important result of the learning by analogy research [Bareiss and Porter 1987; Burstein, 1986; Carbonell, 1983; 1986; Chouraqui, 1982; Gentner, 1983; Kedar-Cabelli, 1985; Kodratoff, 1988; Russel, 1987; Winston, 1986] is that the analogy involves mapping some underlying causal network of relations between analogous situations. The idea is that similar causes are expected to have similar effects.

In DISCIPLE, the explanation of a problem-solving operation may be regarded as a cause for performing the operation. Therefore, two similar explanations are supposed to 'cause' similar problem-solving episodes. Moreover, the explanations are considered to be similar if they are both less general than an overgeneralized explanation that is taken as the analogy criterion.

Figure 19–16 contains an example of such an analogy. The fact that the 'mowicoll' glues both the 'ring' and the 'chassis-membrane-assembly' 'CAUSED' the reduction of the problem of attaching the 'ring' to the 'chassis-membrane-assembly' to a process of gluing them with 'mowicoll'. Because the 'neoprene' glues both the 'screening-cap' and the 'loudspeaker' we may expect (reasoning by analogy) to be able to reduce the problem of attaching the 'screening-cap' and the 'loudspeaker' to a process of gluing them with 'neoprene'.

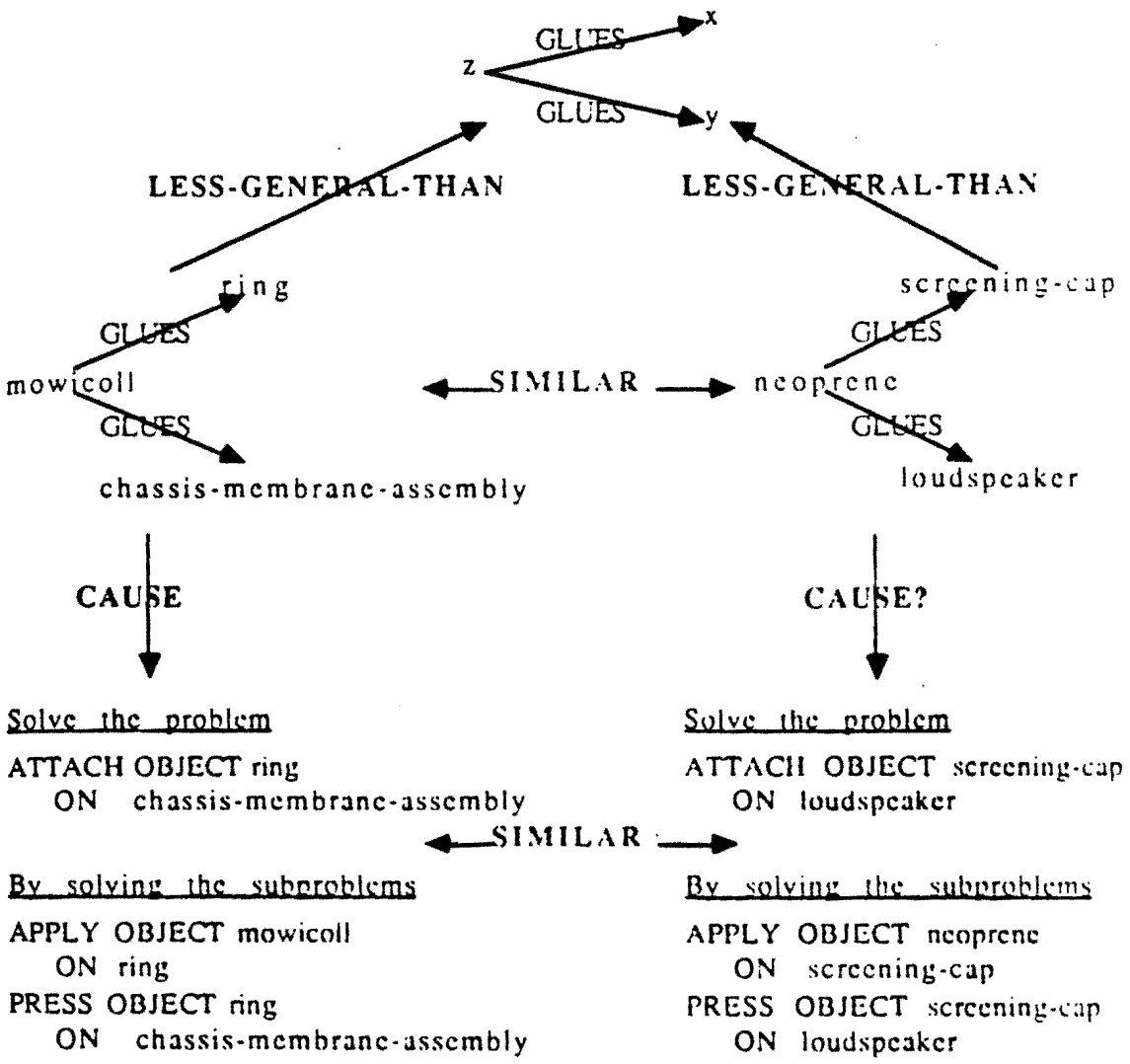


Figure 19–16: An example of analogy

According to the structure-mapping theory of Gentner (1983), analogy usually involves mapping higher order relations (as the 'CAUSE' relation, in our case). Looking for an explanation in terms of relations between objects, DISCIPLE ensures that the 'CAUSE' relation, which it imports by analogy, is a higher order relation.

19.5.4.3 Determining a Reduced Version Space for the Rule to Be Learned

The purpose of the previous sections was to justify the following procedure for determining a reduced version space for the rule to be learned.

First of all DISCIPLE overgeneralizes Example 1 by turning all the objects into variables, thus obtaining:

General Rule 1:
solve the problem
ATTACH OBJECT x ON y
by solving the subproblems
APPLY OBJECT z ON x
PRESS OBJECT x ON y

Next Explanation 1 is rewritten as a lower bound of the applicability condition of General Rule 1 (S bound in Figure 19–17). Notice that it is indeed a lower bound because it reduces General Rule 1 to Example 1, which is known to be true. Further, DISCIPLE determines an analogy criterion that will allow it to generate instances analogous to Example 1.

The analogy criterion is a generalization of Explanation 1. In the case of our example, it was obtained by simply transforming the constants of Explanation 1 into variables, or, if we consider the form of Explanation 1 in Figure 19–17, by dropping the 'ISA' predicates.

In general, *the analogy criterion should be the most general generalization of Explanation 1 that may still be accepted by the user as an explanation of General Rule1*. The analogy criterion *is taken by DISCIPLE* as an upper bound for the applicability condition of General Rule 1 (G bound in Figure 19–17). Thus, the analogy criterion, Explanation 1, and General Rule 1 define a reduced version space [Mitchell, 1978] for the rule to be learned.

IF

G:upper bound (analogy criterion)
(z GLUES x) & (z GLUES y)

S:lower bound (Explanation 1)
(x ISA ring) & (y ISA chassis-membrane-assembly) & (z ISA mowicoll)
& (z GLUES x) & (z GLUES y)

THEN
General Rule 1:
solve the problem
ATTACH OBJECT x ON y
by solving the subproblems
APPLY OBJECT z ON x
PRESS OBJECT x ON y

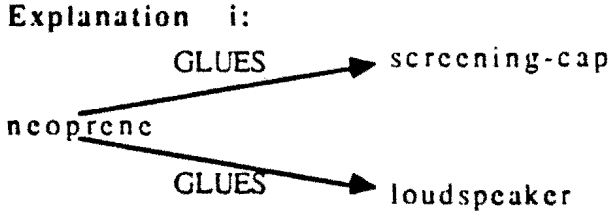
Figure 19–17: A reduced version space for the rule to be learned

Each rule in this space has an applicability condition that is less general than the analogy criterion and more general than Explanation 1. Therefore, it covers only instances that are analogous to Example 1.

19.5.4.4 Generation of Instances

To search the rule in the space from Figure 19–17, DISCIPLE needs positive and negative instances of it. *These instances may be provided by future problem-solving episodes or may be generated by the system itself.*

To generate an instance, DISCIPLE looks in the knowledge base for objects satisfying the analogy criterion. The objects 'screening-cap', 'loudspeaker', and 'neoprene' are such objects. DISCIPLE calls **Explanation-i** the properties of these objects that were used to prove that they satisfy the analogy criterion:



It uses the found objects to generate an instance of General Rule 1 (see Figure 19–17) and asks the user to validate it (see Figure 19–18).

19.5.5 Empirical Learning Mode

The instances generated in the analogy mode are accepted or rejected by the user, being thus characterized as positive examples or as negative examples of the rule to be learned. These instances are used to search the rule in the version space from Figure 19–17.

19.5.5.1 The Use of the Positive Examples

Each positive example shows a true explanation. All these explanations are generalized [Kodratoff and Ganascia, 1986], and the obtained generalization is used as a new lower bound of the condition version space.

May I solve the problem
ATTACH OBJECT screening-cap ON loudspeaker

by solving the subproblems
APPLY OBJECT neoprene ON screening-cap
PRESS OBJECT screening-cap ON loudspeaker ? *Yes*

Figure 19–18: An instance generated by analogy with Example 1

Let us suppose, for instance, that the user accepts the decomposition in Figure 19–18. Then, Explanation-i, computed in Section 19.5.4.4, is a true explanation that may also be rewritten as a lower bound for the applicability condition of General Rule 1:

Explanation i:

(x ISA screening-cap) & (y ISA loudspeaker) & (z ISA neoprene) &
(z GLUES x) & (z GLUES y)

Therefore, DISCIPLE computes a maximally specific, common generalization of the lower bound in Figure 19–17 and of Explanation-i and takes it as a new lower bound of the condition to be learned:

IF

G:upper bound

(z GLUES x) & (z GLUES y)

S:lower bound

(x TYPE solid) & (y TYPE solid) & (z ISA adhesive) &
(z GLUES x) & (z GLUES y)

THEN

General Rule 1:

solve the problem

ATTACH OBJECT x ON y

by solving the subproblems

APPLY OBJECT z ON x
PRESS OBJECT x ON y

Notice that DISCIPLE generalized '(z ISA mowicoll)' and '(z ISA neoprene)' to '(z ISA adhesive)', by applying the well-known rule of climbing the generalization hierarchies [Michalski, 1983]. But it generalized '(x ISA ring)' and '(x ISA screening-cap)' to '(x TYPE solid)' because there is no common generalization of 'ring' and 'screening-cap', and the only relevant property common to 'ring' and 'screening-cap' is that they are both 'solid'. Another common property of 'ring' and 'screening-cap' is that they are both PART-OF 'loudspeaker'. DISCIPLE considers that this property is not relevant because it was not accepted as explanation of Example 1 (see Section 19.5.3.2).

Notice also that the new lower bound is always more specific than the upper bound because both the previous lower bound and Explanation i are less general than the upper bound. However, the generalization of the lower bound was made in the context of an incomplete knowledge. Therefore it could be an overgeneralization, to be later particularized when new knowledge becomes available.

May I solve the problem
ATTACH OBJECT screening-cap ON loudspeaker

by solving the subproblems
APPLY OBJECT scotch-tape ON screening-cap
PRESS OBJECT screening-cap ON loudspeaker? *No*

Figure 19–19: A negative example of the rule to be learned

19.5.5.2 The Use of the Negative Examples

Each negative example shows the incompleteness of Explanation 1 and of its overgeneralization (the analogy criterion). The explanation of why the instance is a negative example points to the features that were not present in Explanation 1. These new features are used to particularize both bounds of the version space.

Let us consider the objects 'screening-cap', 'loudspeaker' and 'scotch-tape' (an adhesive tape). They also satisfy the analogy criterion (the upper bound of the condition version space) but the corresponding instance is rejected by the user (see Figure 19–19).

In this case, DISCIPLE looks for an explanation of the failure because this explanation points to the important object features that were not contained in Explanation 1. The explanation is that 'scotch-tape' is not fluid (therefore, it might not be applied on a curved surface):

Failure Explanation: NOT (scotch-tape TYPE fluid)

That is, the concept represented by 'z' must be fluid. Therefore, DISCIPLE will specialize both bounds of the version space by adding the '(z TYPE fluid)':

IF

G:upper bound
(z GLUES x) & (z GLUES y) & (z TYPE fluid)

S:lower bound
(x TYPE solid) & (y TYPE solid) & (z ISA adhesive) &
(z GLUES x) & (z GLUES y) & (z TYPE fluid)

THEN
General Rule 1:
solve the problem
ATTACH OBJECT x ON y

by solving the subproblems

APPLY OBJECT z ON x

PRESS OBJECT x ON y

In another situation, failing to glue two objects whose surfaces do not fit each other, DISCIPLE discovers the condition that the objects should partially fit:

IF

G:upper bound

(z GLUES x) & (z GLUES y) & (z TYPE fluid) &
(x PARTIALLY-FITS y)

S:lower bound

(x TYPE solid) & (y TYPE solid) & (z ISA adhesive) &
(z GLUES x) & (z GLUES y) & (z TYPE fluid) &
(x PARTIALLY-FITS y)

THEN

General Rule 1:

solve the problem

ATTACH OBJECT x ON y

by solving the subproblems

APPLY OBJECT z ON x

PRESS OBJECT x ON y

The learning process decreases the distance between the two bounds of the version space. This process should, in principle, continue until the lower bound becomes identical with the upper one.

In our case, other negative examples will show that

(x TYPE solid) & (y TYPE solid) & (z ISA adhesive)

are necessary features of the objects 'x', 'y', and 'z'. Thus one learns the rule in Figure 19–6.

However, since the domain theory is weak, we should expect that this will not always happen. Therefore, we will be forced to preserve two conditions (the upper bound and the lower bound), instead of a single applicability condition. We propose to define such a case as being typical of an *uncertain explanation* (in which uncertainty is not expressed by numerical means).

19.5.5.3 Active Experimentation

In the analogy-based mode DISCIPLE may generate many instances of the rule to be learned. However, they are not equally useful for searching the version space.

Therefore, in the empirical learning mode, DISCIPLE will determine the features of the most useful instances, asking for the generation of such instances. Its strategy is to generalize the lower bound of the version space by generalizing the referred objects (i.e., 'mowicoll', 'ring', and 'chassis-membrane-assembly'). It will therefore try to climb the generalization hierarchy of these objects in such a way as to preserve consistency with the necessary condition. During this generalization process, several situations may occur:

- there are different ways to generalize;
- the generalization may cover objects that are not guaranteed to produce positive examples of the rule.

When faced with such problems, DISCIPLE will ask the user "clever" questions (as, for instance, in [Sammut and Banerji, 1986]) whose answers allow it to take the right decision. This process is illustrated in [Tecuci, 1988].

19.5.6 Developing the Domain Theory

As has been shown in Section 19.5.3.2, DISCIPLE looks for explanations in its knowledge base. Because the domain theory is weak, we may expect that it will not always contain the right pieces of explanations. In such situations the pieces of the explanation must be provided by the user.

Let us consider, for instance, that the explanation of the failure in Figure 19–19 was provided by the user. In this case the domain theory will be enriched by storing this explanation: 'NOT (scotch-tape TYPE fluid)'.

More significantly, as a consequence of updating the Lower Bound of the version space, the following relations between the objects that previously generated positive examples of the rule (and are therefore supposed to satisfy the Lower Bound) are added to the domain theory:

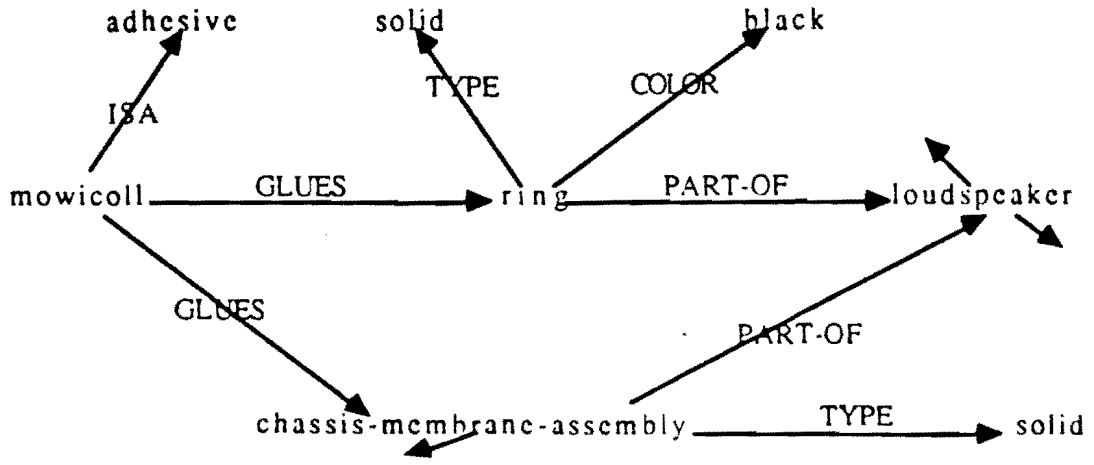
(mowicoll TYPE fluid) & (neoprene TYPE fluid).

19.6 LEARNING IN AN INCOMPLETE THEORY DOMAIN

19.6.1 A Sample of an Incomplete Theory

In the case of DISCIPLE, an incomplete theory of a domain may lack some object descriptions, inference rules, or action models. Also, it may contain incomplete descriptions of these.

An incomplete description of an object lacks certain properties or relations with other objects; an incomplete action model lacks some precondition predicates or some effect predicates; and an incomplete inference rule lacks some left-hand side or right-hand side predicates.



$\forall x\ \forall y\ [(x \text{ GLUED-ON } y) \Rightarrow (x \text{ ATTACHED-ON } y)]$

$\forall x\ \forall y\ \forall z\ [(z \text{ ISA adhesive}) \& (z \text{ GLUES } x) \& (z \text{ GLUES } y) \& (z \text{ BETWEEN } x\ y) \Rightarrow (x \text{ GLUED-ON } y)]$

$\forall x\ \forall y\ [(x \text{ GLUES } y) \Rightarrow (x \text{ ADHERENT-ON } y)]$

Figure 19–20: Incomplete descriptions of the objects from Example 1

Action	Preconditions	Effects
ATTACH OBJECT x ON y	(x TYPE solid) & (y TYPE solid)	(x ATTACHED-ON y)
APPLY OBJECT z ON x	(z ADHERENT-ON x) & (x TYPE solid)	(z APPLIED-ON x)

Figure 19–21: Incomplete models of two actions from Example 1

A sample of an incomplete theory about Example 1 (Figure 19–5) is given in the Figures 19–20 and 19–21.

As one may notice, the explicit properties and relations of the objects 'ring', 'chassis-membrane-assembly' and 'mowicoll' are the ones considered in the case of the weak theory (see Figure 19–13).

Also notice that this incomplete theory lacks entirely the model of the action 'PRESS'. It also contains an incomplete model of the action 'APPLY', model lacking the precondition predicate '(z TYPE fluid)'.

19.6.2 General Presentation of the Learning Method

In this case, the learning method combines the two learning methods presented previously. First, the system will construct an incomplete proof of Example 1 and will generalize it, as in a complete theory. In this way, it will determine an *over-*

generalized explanation of Example 1. Then, the system will use the overgeneralized explanation as an *analogy criterion* to perform experiments and to synthesize the general rule, as in a weak theory:

1. Prove that the solution indicated by the user is indeed a solution of the problem to solve. Because the domain theory is incomplete, the system may ask the user focused questions in order to fill the possible gaps in the proof. The leaves of the proof tree represents an incomplete explanation of Example 1.
2. If the user's solution contains new actions, then use the proof found in step 1 in order to define initial version spaces for the models of these actions. As a side effect of rule learning, DISCIPLE will learn the models of these new actions.
3. Overgeneralize the proof tree found in step 1, as in a complete theory. If an action model is incompletely learned then use the upper bound of its preconditions and effects. The leaves of the overgeneralized proof tree represent an overgeneralized explanation of Example 1, being taken by DISCIPLE as an analogy criterion.
4. Formulate a reduced version space for the rule to be learned, as in a weak theory, by using the explanation found in step 1 and the overgeneralized explanation found in step 3.
5. Search the rule in the version space defined in step 4 by performing experiments, as in a weak theory. Use the overgeneralized proof determined in step 3 in order to find the explanations of the failures.

In the next section we shall illustrate this learning method.

19.6.3 Incomplete Proving of the Example

Even when the objects, the inference rules, and the actions are incompletely specified, one may be able to construct a proof tree, which lacks some parts of the complete proof tree (see Chapter 18, this volume and [Wilkins, 1988]).

When the system lacks inference rules or action models, it will try to sketch the proof tree both top-down and bottom-up, and will ask the user focused questions, in order to connect the different parts of the proof.

Using the incomplete theory about Example 1, presented in the previous section, the system may build the following proof of Example 1 (see Figure 19–22). The dotted lines from the proof tree do not result from the domain theory but are hypotheses made by the system and confirmed by the user. For instance, the system makes the hypothesis that

(mowicoll BETWEEN ring chassis-membrane-assembly)

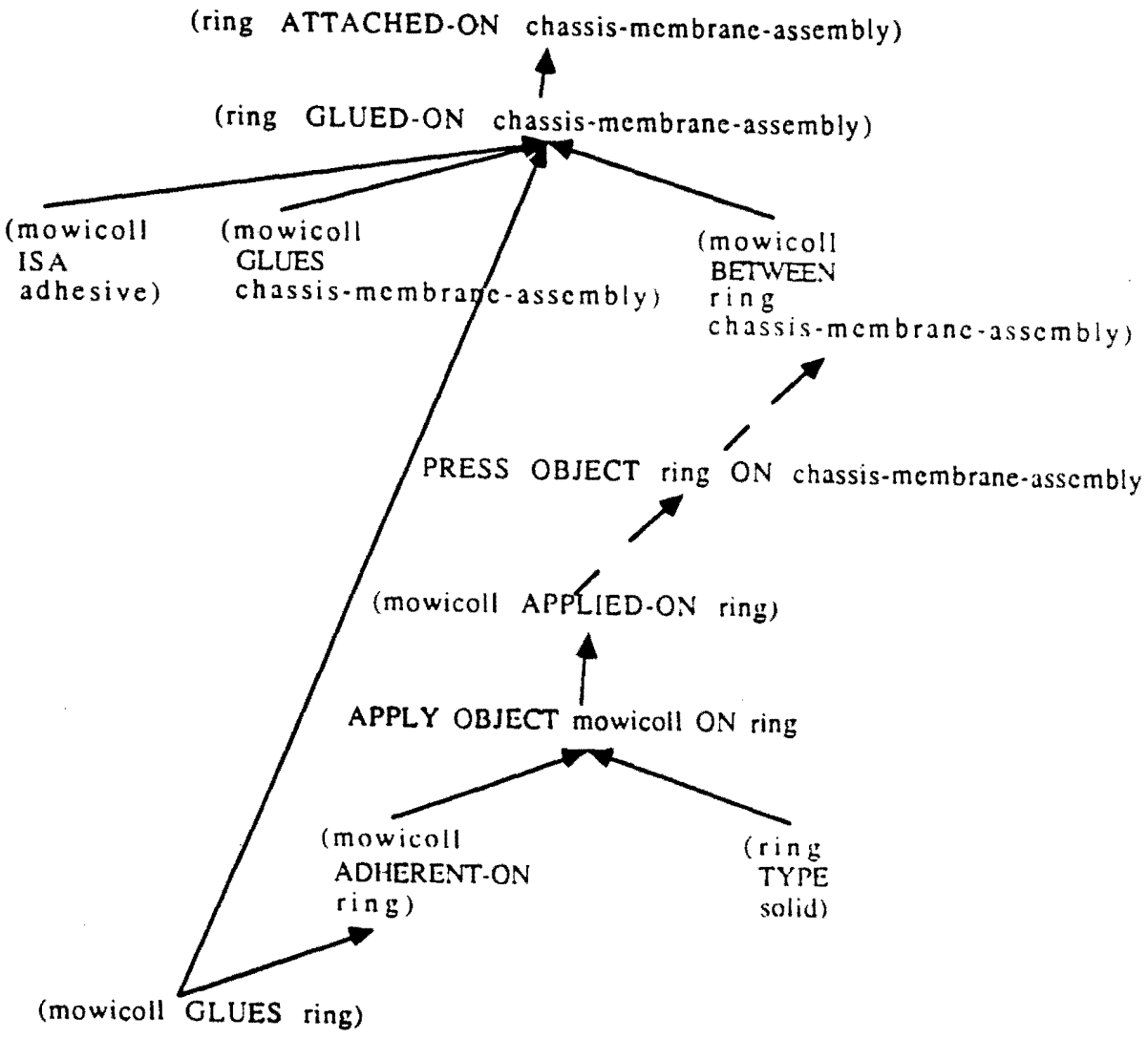


Figure 19–22: An incomplete proof of Example 1

is an effect of the action

PRESS OBJECT ring ON chassis-membrane-assembly

from the fact that all the other left-hand side literals of the inference rule

$$\forall x\ \forall y\ \forall z\ [(z \text{ ISA adhesive}) \ \&\ (z \text{ GLUES } x) \ \&\ (z \text{ GLUES } y) \ \&\ (z \text{ BETWEEN } x\ y) \Rightarrow (x \text{ GLUED-ON } y)]$$

are true in the current situation, that is

$$[(\text{mowicoll ISA adhesive}) \ \&\ (\text{mowicoll GLUES ring}) \ \&\ (\text{mowicoll GLUES chassis-membrane-assembly})] = \text{TRUE}$$

and the literal '(mowicoll BETWEEN ring chassis-membrane-assembly)' is not known to be true.

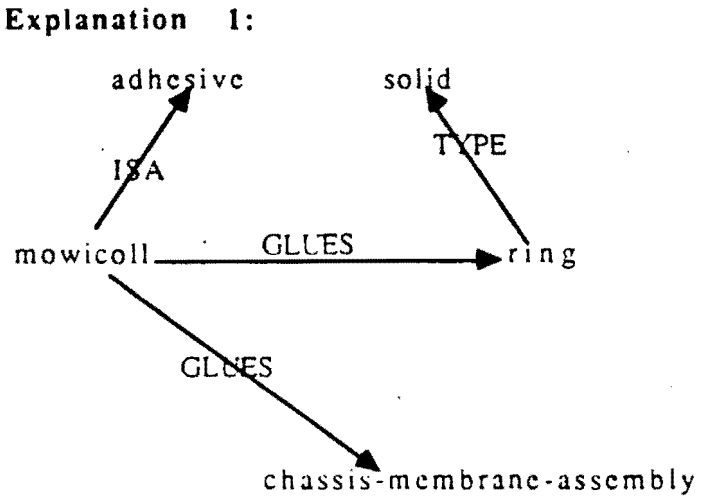


Figure 19–23: The relevant features of Example 1, revealed by the proof tree in Figure 19–22

Comparing the proof tree in Figure 19–22 with the complete one in Figure 19–11, one may easily notice that it lacks some leaves. Nevertheless, the leaves that are present represent some important features of the objects from Example 1; features that in the case of a weak theory would correspond to the explanation of Example 1 shown in Figure 19–23.

19.6.4 Defining Version Spaces for the Unknown Actions

The incomplete proof allows one to define initial version spaces for the models of the unknown actions used in the proof. For instance, one may define the following version space for the action 'PRESS':

Action	Preconditions	Effects
PRESS OBJECT x ON y	*upper bound:* (z APPLIED-ON x) *lower bound:* (z APPLIED-ON x) & (x ISA ring) & (y ISA chassis-membrane-assembly) & (z ISA mowicoll)	*upper bound:* (z BETWEEN x y) *lower bound:* (z BETWEEN x y) & (x ISA ring) & (y ISA chassis-membrane-assembly) & (z ISA mowicoll)

The *lower bounds* for the preconditions and effects are taken directly from the proof tree. The *upper bound* of the effects is the generalization of the lower bound (mowicoll BETWEEN ring chassis-membrane-assembly) taken from the premise of the inference rule

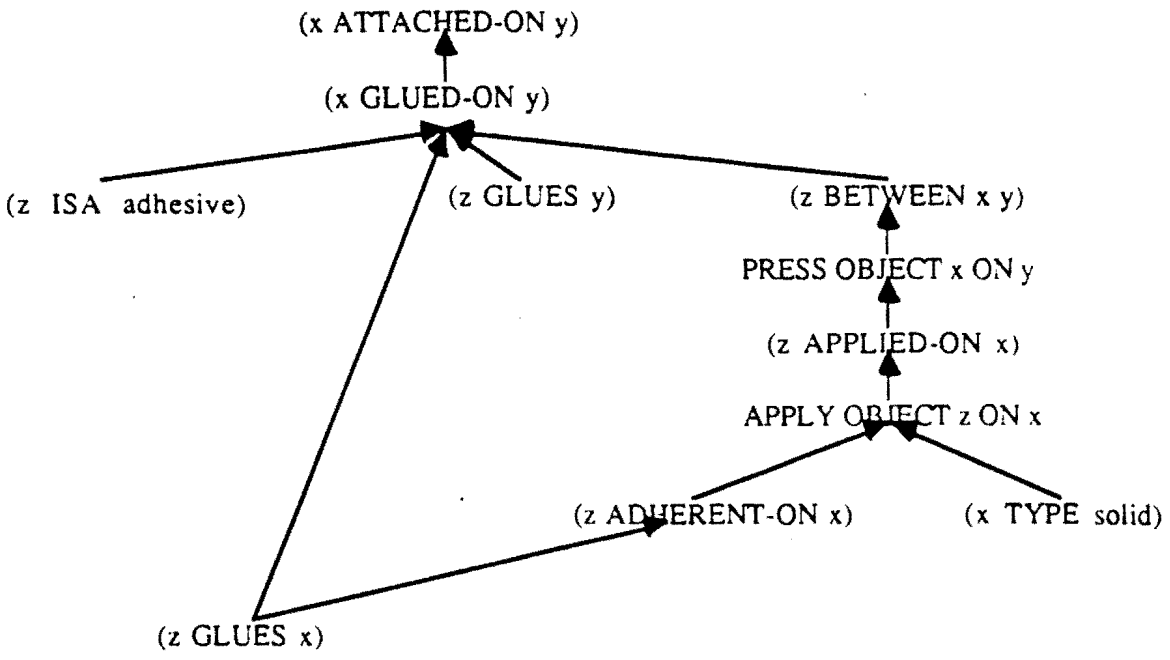


Figure 19–24: The generalization of the proof in Figure 19–22

$$\forall x \forall y \forall z[(z \text{ ISA adhesive}) \ \& \ (z \text{ GLUES } x) \ \& \ (z \text{ GLUES } y) \ \& \ (z \text{ BETWEEN } x \ y) \Rightarrow (x \text{ GLUED-ON } y)]$$

The *upper bound* of the preconditions is the generalization of the lower bound, taken from the effects of the model of the action 'APPLY OBJECT z ON x'. During the learning of the decomposition rule in Figure 19–6, the system will also refine the model of the action 'PRESS'.

19.6.5 Generalization of the Incomplete Proof

Once the proof in Figure 19–22 is built, the system will generalize it, as in a complete theory (see Figure 19–24).

Let us notice that, for generalizing the proof, the system used the upper bounds of the preconditions and effects of the action 'PRESS'.

19.6.6 Determining a Reduced Version Space for the Rule to Be Learned

As in the case of a weak theory, the Explanation 1 in Figure 19–23 may be rewritten as a Lower Bound for the applicability condition of General Rule 1 (Figure 19–25). Also, the leaves of the generalized proof tree in Figure 19–24 provide an *overgeneralized explanation* of Example 1. This overgeneralized explanation corresponds to the *analogy criterion* from a weak theory and is taken by DISCIPLE as an Upper Bound for the applicability condition of General Rule 1 (see Figure 19–25).

IF

G:upper bound (analogy criterion)

(x TYPE solid) & (z ISA adhesive) & (z GLUES x) &
(z GLUES y)

S:lower bound (explanation 1)

(x ISA ring) & (x TYPE solid) &
(y ISA chassis-membrane-assembly) &
(z ISA adhesive) & (z GLUES x) & (z GLUES y)

THEN

General Rule 1:

solve the problem

ATTACH OBJECT x ON y

by solving the subproblems

APPLY OBJECT z ON x

PRESS OBJECT x ON y

Figure 19–25: A reduced version space for the rule to be learned

Therefore, as in a weak theory, the system is able to formulate the following version space for the rule to be learned:

19.6.7 Searching the Rule in the Version Space

As soon as the version space from Figure 19–25 has been determined, rule learning will continue as in a weak theory. This time, however, the generalized proof tree in Figure 19–24 provides a focus for the process of finding the explanations of the failures. To illustrate this, let us consider again the failure in Figure 19–19.

In this case, the system generates the instance of the generalized proof in Figure 19–24, corresponding to this problem-solving episode (by replacing 'x', 'y', and 'z' with 'screening-cap', 'loudspeaker', and 'scotch-tape', respectively).

The fact that the user rejected the solution proposed by the system proves that the leaves of the instantiated tree do not imply the top of the tree (the leaf predicates are true but the top predicate is not).

This means that some action models or inference rules are faulty (incomplete, in our case). To detect them, the system follows the proof tree from bottom up, asking the user to validate each inference step. If the user says that the effect of an action or the consequent of an inference rule is not true, then the corresponding action model (inference rule) may be the incomplete one.

Therefore, in an incomplete theory, finding the explanations of the failures reduces to finding the knowledge that is lacking from the knowledge pieces. In this case, the generalized proof in Figure 19–24 plays the role of a justification structure for the rule to be learned, as in [Smith, *et al.*, 1985].

19.7 EXPERIMENTS WITH DISCIPLE

We have implemented a version of DISCIPLE in Common LISP [Steele, 1984] and we have used it to learn rules in several domains as, for instance, manufacturing, commonsense planning, chemistry, and architecture [Kodratoff and Tecuci, 1987; Tecuci and Kodratoff, 1990].

With a very poor theory of chemistry,[1] DISCIPLE was able to learn, starting from the example of the chemical reaction (NaOH + HCl → H2O + NaCl), that, in general, (Base + Acid → Water + Salt). More precisely, starting from the example

the problem

COMBINE SUBSTANCE1 NaOH SUBSTANCE2 HCl

has the following solution

RESULT SUBSTANCE1 H20 SUBSTANCE2 NaCl

DISCIPLE learned the following rule:

IF

G:upper bound

(b COMPOSED-OF x1) & (b COMPOSED-OF x2) &
(a COMPOSED-OF x3) & (a COMPOSED-OF x4) &
(w ISA water) & (w COMPOSED-OF x1) (COMPOSED-OF x3) &
(s ISA salt) & (s COMPOSED-OF x2) (s COMPOSED-OF x4) &
(s (COMPOSED-OF ANION-OF) a) &
(s (COMPOSED-OF CATION-OF) b))

S:lower bound

(b ISA base) & (b COMPOSED-OF x1) & (b COMPOSED-OF x2) &
(a ISA acid) & (a COMPOSED-OF x3) & (a COMPOSED-OF x4) &
(w ISA H2O) & (w COMPOSED-OF x1) & (COMPOSED-OF x3) &
(s ISA salt) & (s COMPOSED-OF x2) & (s COMPOSED-OF x4) &
(s (COMPOSED-OF ANION-OF) a) &
(s (COMPOSED-OF CATION-OF) b)) &

[1] This application was suggested by D. Sleeman.

(x1 ISA OH) & (x2 ISA METAL) & (x3 ISA H) &
(x4 ISA METALLOID)

THEN

General Rule

the problem

COMBINE SUBSTANCE1 b SUBSTANCE2 a

has the following solution

RESULT SUBSTANCE1 w SUBSTANCE2 s

The lower bound of this rule says that all the positive examples share the structure (Base + Acid → Water + Salt), together with the appropriate components: For instance, x2 is the metal of the base b (because "b COMPOSED-OF x2") that will go to the salt s (because "s COMPOSED-OF x2").

The upper bound of the rule says that none of the negative examples met shared the structure (Something + Something → Water + Salt) together with the appropriate components; i.e., there might be compounds leading to water and salt, other than bases and acids.

The application from architecture[2] consists in designing a building. In this application, DISCIPLE may learn rules to refine specifications of objects. For instance, starting from an example of a door separating a hall from a sleeping room (for which the expert established that it should be opened towards the sleeping room), DISCIPLE learned a general rule for establishing the direction of opening of the doors: "A door separating a house-piece from a room should be opened towards the room."

From these experiments we have learned the following:

- It is fairly easy to define a small initial domain theory for DISCIPLE;
- Although DISCIPLE has knowledge to generate hundreds of examples it actually generates a few of them only in order to learn a rule.

19.8 CONCLUSIONS

Trying to cope with the complexity of the real-world applications, we have made the hypothesis that DISCIPLE's domain theory is nonhomogeneous, describing completely some parts of the domain, but only incompletely or even poorly, the other parts. The use of DISCIPLE is tuned to problem-solving situations in which some variabilization is meaningful. For instance, a set of zeroth-order rules solving a problem will not yield interesting results under DISCIPLE.

---

[2] Suggested by F. Guena and K. Zreik.

We have shown how the system is able to learn the same general rule from the same example, by using a method corresponding to its theory about the example.

In the context of a complete theory, DISCIPLE uses explanation-based learning. It is thus able to learn a justified rule from a single example and may also reject incorrect examples.

The learning method in the context of a weak theory integrates different learning paradigms: explanation based learning, learning by analogy, empirical learning, and learning by questioning the user. Among the most relevant features of this learning method one could mention:

- the notion of "explanation" in a weak theory and a heuristic method to find such explanations,
- the use of analogy to define a reduced version space for the rule to be learned,
- the use of both the explanations of the successes and the explanations of the failures to search the rule in its version space,
- the formulation of "clever" questions in order to extract useful knowledge from the expert,
- the possibility of hiding the learned rules from the expert,
- a great confidence in the human expert.

In the context of an incomplete theory, DISCIPLE learns by combining the method corresponding to the complete theory with the method corresponding to the weak theory. This method borrows features from both the learning method in a complete theory (may reject incorrect examples, learns justified rules) and from the learning method in a weak theory (use of analogy, clever questions to the user, etc.).

It is interesting to notice that, although in each of the presented cases the system learned the same general rule, the effect of this rule on the future behavior of the system depends of the domain theory: In a *complete theory*, the learned rule improves only the *performance* of the system, in a *weak theory* it develops the *competence* of the system, and, in an *incomplete theory*, it may develop both the *performance* and the *competence*.

Let us also notice that, by the integration of these three learning methods, DISCIPLE proposes a solution to the so-called "falling off the knowledge cliff" problem of current systems. This problem is that a system performs well within the scope of the knowledge provided to it, but any slight move outside its narrow competence causes the *performance to deteriorate rapidly* [Michalski, 1986]. On the contrary, in DISCIPLE, the move from one part of the application domain, characterized by a complete theory, to another part, characterized by an incomplete theory or by a weak theory, causes only *a slight deterioration of the performance*, this effect being obtained by a corresponding replacement of the learning method used.

There are also several *weaknesses* of DISCIPLE, on which will shall direct our future research. For instance, the expressions DISCIPLE deals with are made of

predicates, constants, and variables, but no actual function evaluation is going to take place.

A semantic limitation comes from the fact that the *generality of the learned rule is limited by the generality of the overgeneralized explanation* (the analogy criterion), which may not be in the most general form. However, the rule may be further generalized, in response to a problem-solving situation in which the rule does not apply, and the user says that it should apply. In this case, the condition of the rule may be generalized to cover the new situation as well.

Also, the method of finding an explanation in a weak theory is not powerful enough. Other sources of knowledge are needed, as well as metarules for finding far off explanations. One possible extension of the current method is suggested by the way CLINT [De Raedt and Bruynooghe, 1989] changes its description language in order to be able to learn a concept. DISCIPLE might also use an ordered series of explanation schemas E1, E2, ... , En, ... First, it will be looking for an explanation of the form E1 (for instance, a path of length 1 between two concepts). If no such explanation is found, then it will be looking for an explanation of the form E2 (for instance, a path of length 2 between two concepts), and so on.

While DISCIPLE uses control knowledge in the form of metarules [Tecuci, 1988], such knowledge is not learned—it must be provided by the user. Therefore, if two experts provide different solutions to the same problem, DISCIPLE simply generates two different rules. The learning mechanisms of DISCIPLE should be used to propose explanations of this difference and find metaexplanations that can become metapreconditions on the use of the rules.

An important future direction of research consists in developing the learning methods of DISCIPLE in order to deal with other types of imperfections in the domain theory [Mitchell, Keller, and Kedar-Cabelli, 1986; Rajamoney and DeJong, 1987]. We shall consider, for instance, imperfections resulting from the fact that certain pieces of knowledge (objects, inference rules, action models) contain minor errors in their definitions in that parts of these definitions may be either more general or less general than they should be.

A weakness of all the learning apprentice systems is that they need an initial domain theory and provide no means of defining it. Although DISCIPLE facilitates this task by accepting a nonhomogeneous domain theory, the task remains difficult. A solution here is that proposed by BLIP [Morik, 1989], which is an interactive learning system mainly concerned with the construction of a domain theory as a first phase of building a knowledge-based system. Therefore, a very promising research direction seems that of building a system incorporating the capabilities of BLIP and DISCIPLE. Such a system could be an effective tool for building knowledge-based systems:

- In the first stage, the system and the user will build together an initial theory of the application domain. This theory, containing elementary knowledge about

the domain (basic concepts and inference rules), will be neither complete nor entirely correct;

- In the second stage, the system and the user solve problems together and, during this cooperative problem solving, the system will learn general problem-solving rules. Since this learning takes place in the context of an imperfect domain theory, the learned rules will accumulate exceptions. These exceptions correspond in fact to lacking concepts in the domain theory. When too many exceptions are accumulated, the domain theory has to be refined. Therefore, one reenters the first stage and reformulates the domain theory to better characterize the current knowledge of the system.

There are also several *lessons* we have learned from the design of DISCIPLE. One is *to cope with the complexity of real-world applications, one should use any available learning technique*. Indeed, the different learning paradigms have many complementary prerequisites and effects. Therefore they may be synergistically combined.

Another lesson is that *full formalization of imperfect theories is short-time harmful*. Indeed, forcing the expert to completely formalize a domain theory (which may not even have such a complete theory) may result in a degradation of the knowledge he provides.

Last, we have discovered that *overgeneralization is not only harmless, but also useful and necessary when interacting with a user*, allowing the identification of features usually neglected by the expert.

ACKNOWLEDGMENTS

This work has been sponsored by PRC-GRECO "Intelligence Artificielle" and the Romanian CNST. The chapter was written while one of the authors was at LRI, on leave from his institute. His expenses were taken in charge through an agreement between the French CNRS and the Romanian Academy of Sciences. We wish to express our gratitude to all these institutions. We also wish to thank Mr. Zani Bodnaru, for his indispensable contribution as domain expert.

References

Bareiss, E. and Porter, B. 1987. PROTOS: An Exemplar-based Learning Apprentice, in Langley, P. (ed), *Proc. Fourth Int. Workshop on Machine Learning*, Irvine.

Burstein, M.H. 1986. Concept Formation by Analogical Reasoning and Debugging, in Michalski, R., Carbonell, J. and Mitchell, T. (eds), *Machine Learning: An Artificial Intelligence Approach, Volume II*, Morgan Kaufmann, pp. 351–370.

Carbonell, J. 1986. Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition, in Michalski, R., Carbonell, J. and Mitchell, T. (eds), *Machine Learning: An Artificial Intelligence Approach, Volume II*, Morgan Kaufmann, pp. 371–392.

Carbonell, J. and Gil, Y. 1987. Learning by Experimentation, in Langley, P. (ed), *Proc. Fourth Int. Workshop on Machine Learning*, Irvine, Morgan Kaufmann.

Chouraqui, E. 1982. Construction of a Model for Reasoning by Analogy, in *Proceedings of the European Conference on Artificial Intelligence*, Orsay, France.

DeJong, G. and Mooney, R. 1986. Explanation-based Learning: An Alternative View, in *Machine Learning*, 1, pp. 145–176.

De Raedt, L. and Bruynooghe, M. 1989. Towards Friendly Concept-Learners, in *Proceedings of IJCAI-89*, Detroit, Morgan Kaufmann.

Fikes, R.E., Hart, P.E. and Nilsson, N.J. 1972. Learning and Executing Generalized Robot Plans, *Artificial Intelligence*, 3, pp. 251–288.

Gentner, D. 1983. Structure-Mapping: a Theoretical Framework for Analogy, *Cognitive Science*, 7, pp.155–170.

Kedar-Cabelli, S. 1985. Purpose-directed Analogy, in *Proceedings of the Cognitive Science Society*, pp. 150–159, Irvine.

Kodratoff, Y. 1988. *Introduction to Machine Learning*, Pitman, London. Available in the U.S. from Morgan Kaufmann.

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

Kodratoff, Y. and Tecuci, G. 1987. Techniques of Design and DISCIPLE Learning Apprentice, *International Journal of Expert Systems: Research and Applications*, vol.1, no.1, pp. 39–66.

——— , 1989. The Central Role of Explanations in DISCIPLE, in Morik, K. (ed), *Knowledge Representation and Organization in Machine Learning*, Springer-Verlag, Berlin.

Michalski, R.S. 1983. A Theory and a Methodology of Inductive Learning, *Artificial Intelligence* 20, pp. 111–161.

——— , 1984. Inductive Learning as Rule-Guided Transformation of Symbolic Descriptions: A Theory and Implementation, in Biermann, A.W., Guiho, G., and

Kodratoff, Y. (eds), *Automatic Program Construction Techniques*, Macmillan Publishing Company, pp. 517–552.

——— , 1986. Understanding the Nature of Learning: Issues and Research Directions, in Michalski, R., Carbonell, J. and Mitchell, T. (eds), *Machine Learning: An Artificial Intelligence Approach, Volume II*, Morgan Kaufmann, pp. 3–25.

Mitchell, T.M. 1978. *Version Spaces: An Approach to Concept Learning*, Doctoral dissertation, Stanford University.

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

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

Mooney, R. and Bennet, S. 1986. A Domain Independent Explanation Based Generalizer, in *Proceedings AAAI-86*, Philadelphia, pp. 551–555.

Morik, K. 1989. Sloppy modeling, in Morik, K. (ed), *Knowledge Representation and Organization in Machine Learning*, Springer Verlag, Berlin.

Pazzani, M.J. 1988. Integrating Explanation-based and Empirical Learning Methods in OCCAM, in Sleeman, D. (ed), *Proceedings of the Third European Working Session on Learning*, Glasgow.

Quillian, M.R. 1968. Semantic Memory, in Minsky, M., (ed), *Semantic Information Processing*, MIT Press, Cambridge, MA, pp. 227–270.

Rajamoney, S. and DeJong, G. 1987. The Classification, Detection and Handling of Imperfect Theory Problems, *Proc. IJCAI-87*, Milan, pp. 205–207.

Russel, S.J. 1987. Analogy and Single-Instance Generalization, in Langley, P. (ed) *Proc. Fourth Int. Workshop on Machine Learning*, Irvine, Morgan Kaufmann.

Sammut, C. and Banerji, R.B. 1986. Learning concepts by asking questions, in Michalski, R.S., Carbonell, J.G., Mitchell, T.M. (eds), *Machine Learning: An Artificial Intelligence Approach, Volume II*, Morgan Kaufmann, pp. 167–191.

Smith, R., Winston, H., Mitchell, T. and Buchanan, B. 1985. Representation and Use of Explicit Justifications for Knowledge Base Refinement, *Proc. IJCAI-85*, Los Angeles, Morgan Kaufmann.

Sridharan, N. and Bresina, J. 1983. *A Mechanism for the Management of Partial and Indefinite Descriptions*, Technical Report CBM-TR-134, Rutgers Univ.

Steele, G. 1984. *COMMON LISP: The Language*, Digital Press.

Tate, A. 1977. Generating Project Networks, *Proc. IJCAI-77*, Massachusetts, Morgan Kaufmann, pp. 888–893.

Tecuci, G. 1988. *DISCIPLE: A Theory, Methodology, and System for Learning Expert Knowledge*, PhD Thesis, University of Paris-Sud, 1988.

Tecuci, G. and Kodratoff, Y. 1990. How to learn it with DISCIPLE, LRI Research Report, Orsay.

Tecuci, G., Kodratoff, Y., Bodnaru, Z., and Brunet, T. 1987. DISCIPLE: An expert and learning system, Expert Systems 87, Brighton, December, 14–17, in Moralee, D.S., (ed), *Research and Development in Expert Systems IV*, Cambridge University Press.

Wilkins, D.C. 1988. Knowledge Base Refinement Using Apprenticeship Learning Techniques, *Proceedings AAAI-88*.

Winston, P.H. 1986. Learning by Augmenting Rules and Accumulating Censors, in Michalski, R.S., Carbonell, J.G., Mitchell, T.M. (eds), *Machine Learning: An Artificial Intelligence Approach, Volume II*, Morgan Kaufmann, pp. 45–61.