ORSAY
no d'ordre: 647

UNIVERSITE DE PARIS-SUD

CENTRE D'ORSAY

T H E S E

présentée

Pour obtenir

Le titre de DOCTEUR en SCIENCE

PAR

Gheorghe TECUCI

SUJET

**DISCIPLE: A Theory, Methodology and System
for Learning Expert Knowledge**

soutenue le 7 juillet 1988 devant la Commission d'examen

For Sanda and my father

Acknowledgements

I am very grateful to Yves Kodratoff for encouraging to pursue research in this fascinating realm of learning, for his generous support, for directing my work with so much competence, and for all these years of joint research.

I thank Joffroy Beauquier for honoring me by agreeing to chair the thesis commission, Françoise Fogelman and Alexandre Parodi for reporting on my thesis, and Jean-Gabriel Ganascia for his friendship and his good advice.

The members of the "Learning and Inference" team have also helped me. I want to thank them all, and especially Antoine Cornuejols, Jean-Jacques Cannat, Thierry Conrad, Luc De Raedt, Martha Franova, Michel Manago, Céline Rouveirol and Stephen Thorp. I thank all the members of LRI for their hospitality.

I express my gratitude to Mihai Draganescu who constantly encouraged and helped me, who suggested writing this thesis to me, and who is for me a human model to follow.

I am also indebted to Mircea Petrescu for all I learned from him, for his encouragement, and for offering me his time with such great generosity.

I thank Margareta Draghici for her trust in my work and for giving me the means of pursuing the research reported in this thesis.

Zani Bodnaru acted admirably as the domain expert of DISCIPLE.

Dan Tufis was always available when I needed advice or help.

This work has been sponsored by the Romanian CNST and it was greatly facilitated by several research visits to the LRI. My expenses have been taken in charge through an agreement between the French CNRS and the Romanian Academy of Sciences. I wish to express my gratitude to these institutions, as well as to Nicolae Badea for his help, his confidence in my work, and the support given to fundamental research.

None of this work would have been possible without the constant encouragement, support, and help of my wife Sanda. I hope she will find here the expression of my love and gratitude.

DISCIPLE: a Theory, Methodology and System for Learning Expert Knowledge

Gheorghe TECUCI
Research Institute for Computers and Informatics
71316, Bd. Miciurin 8-10, Bucharest 1, ROMANIA

Abstract

This thesis presents DISCIPLE, a system illustrating a theory and a methodology for learning expert knowledge. DISCIPLE integrates a learning system and an empty expert system, both using the same knowledge base. It is provided with an initial 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 of 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, and empirical learning, 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.

Keywords: Learning, Expert Systems, Artificial Intelligence, Knowledge Acquisition, Explanation-Based Learning, Learning by Analogy, Empirical Learning, Conceptual Clustering

C O N T E N T

A P P E N D I X

1. INTRODUCTION

The present success of Artificial Intelligence is mostly due to the knowledge-based systems which 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.

This problem is largely recognized as the "knowledge acquisition bottleneck" of the knowledge-based systems ([Feigenbaum, 1977], [Michalski, 1986]).

Recent Machine Learning achievements ([Michalski, Carbonell & Mitchell, 1983, 1986], [Langley, 1987], [Laird, 1988]) offer new solutions to the knowledge acquisition problem and open a new area in the evolution of expert systems, that is, Expert Systems capable to automatically acquire knowledge and learn.

The Learning Apprentice Systems (LAS) are examples of such learning expert systems.

A Learning Apprentice System is an interactive knowledge-based consultant which 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 & al. 1985].

Representative examples of this approach are the systems GENESIS [Mooney & DeJong, 1985] and LEAP [Mitchell & al. 1985]. The domain of expertise of LEAP is the VLSI design and that of GENESIS is story understanding.

A common feature of LEAP and GENESIS is that they are based on a strong (complete) domain theory. Such a complete theory allows them to learn a general rule or schemata from a single example.

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 apprentice 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.

The goal of this thesis was to develop a theory and methodology of expert knowledge acquisition in such nonhomogeneous domain theories.

The system illustrating this theory and methodology is called DISCIPLE.

DISCIPLE is an interactive system which integrates an empty expert system and a learning system. It is initially provided with elementary knowledge about an application domain (knowledge representing a nonhomogeneous theory of the domain) 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 is based on problem reduction. 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 (i.e. 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.

The knowledge base of DISCIPLE contains object descriptions, action models, problem reduction rules, and meta-rules.

The object descriptions and the action models are elementary knowledge about an application domain.

Using such elementary knowledge, DISCIPLE learns general problem reduction rules from examples of reductions provided by its user, during the normal use of the system.

The meta-rules, for choosing between the rules applicable to reduce a problem, have to be defined by the user, once such competing rules have been learned.

The method of learning a general problem solving rule from a particular solution indicated by the user (which constitutes an example of the rule) depends on the system's theory (knowledge) about the solution.

We have considered three types of theories: complete, weak, and incomplete.

A complete theory about an example consists of the complete descriptions of all the objects and actions contained in the example.

A weak theory about an example consists only of incomplete descriptions of the objects from this example.

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 the example. Also, it may lack some object descriptions or action models.

In the case of a complete theory about the example, the learning method of DISCIPLE follows the explanation-based learning paradigm.

First, DISCIPLE proves that the solution indicated by the user is indeed a solution of the problem to solve. Then it generalizes the proof tree as much as possible so that the proof remains valid, and formulates the learned rule from this generalized proof.

In the case of a weak theory about the solution indicated by the user, DISCIPLE learns interactively by synergistically combining explanation-based learning, learning by analogy, and empirical learning.

First DISCIPLE looks for a shallow explanation of 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 which are analogous with the user's solution. DISCIPLE carefully generates such 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.

In the case of an incomplete theory about the user's solution, DISCIPLE learns a general rule by combining the learning methods mentioned previously.

First, the system will construct an incomplete proof of the user's solution and will generalize it, as in a complete theory. This will allow DISCIPLE to define a reduced version space for the rule to be learned and to perform experiments, as in a weak theory.

It is interesting to notice that the effect of the learned rule on the future behavior of the system depends of the domain theory. In a complete theory, the learned rule improves the performance of the system, in a weak theory it develops its competence and, in an incomplete theory, it develops both its performance and competence.

Another effect of learning in the context of a weak theory or an incomplete theory is that of developing the domain theory.

2. AN INTUITIVE VIEW OF DISCIPLE

2.1 The overall architecture of DISCIPLE

DISCIPLE illustrates an approach to the knowledge transfer from a human expert to an expert system.

We would like to allow the expert to introduce into the knowledge base of the system only that knowledge which he may easily formalize, and to enable the system to learn the rest of the necessary knowledge.

The overall architecture of DISCIPLE is presented in figure 2.1.

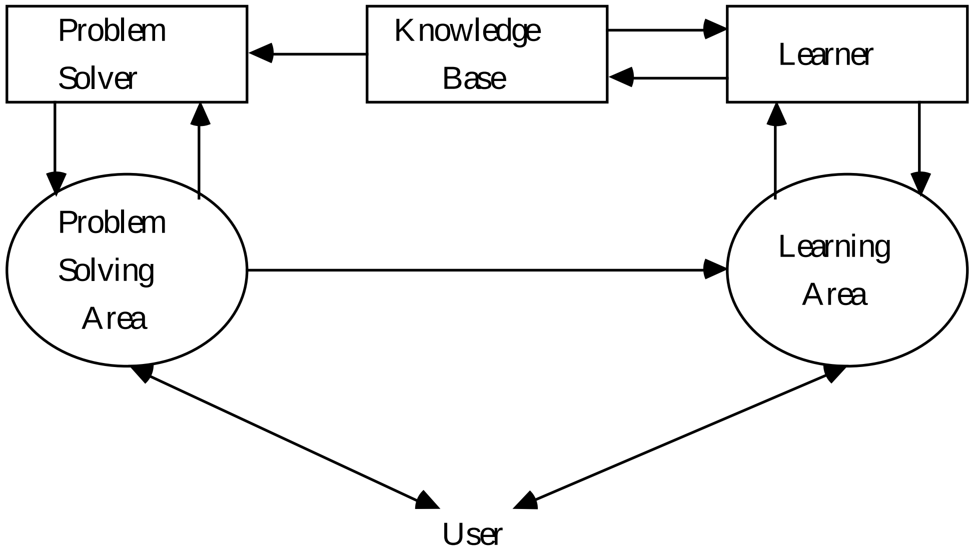


Figure 2.1. Overall architecture of DISCIPLE.
The arrows indicate the main directions of information flow.

DISCIPLE is an interactive system which integrates an empty expert system and a learning system.

Initially, the human expert has to introduce into the knowledge base of the system a theory of the application domain, theory consisting of elementary knowledge about the domain.

Next, DISCIPLE may be used to interactively solve problems, 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 all the problem solving steps. The user may agree or reject them. Therefore, during the course of its functioning as an Expert System, DISCIPLE may encounter two situations.

Either the current problem-solving step (which we shall call partial solution) is accepted by the user. Then, the current state of the knowledge base is judged as satisfactory, and no learning will take place.

Or it is unable to propose any partial solution (or the solution it proposes is rejected by the user). Then, 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 with 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 helpful assistant in problem solving to a genuine expert.

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.

Therefore, the task of the learning system is that of learning general problem reduction rules from examples.

2.2 DISCIPLE as an Expert System

2.2.1 Problems to solve with DISCIPLE

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 NOAH [Sacerdoti, 1975], NONLIN [Tate, 1977], HILARE [Giralt & al. 1979], MOLGEN [Stefik, 1980], PLANX10 [Sridharan & Bresina, 1982], SIPE [Wilkins, 1984], and others.

An example of such a problem is the following one:
- given the incomplete specifications of a loudspeaker;
- design the actions needed to manufacture the loudspeaker.

This problem is solved by a successive decomposition of the complex operation of manufacturing the loudspeaker into simpler operations, and better defining these simpler operations by choosing tools, materials, or verifiers, which are in turn successively refined.

In this process, DISCIPLE will have to completely design the loudspeaker, as well as the tools needed to manufacture it.

2.2.2 Elementary knowledge about an application domain

In order to be able to build a manufacturing technique for a loudspeaker, DISCIPLE needs various types of knowledge:

- knowledge about the components of the loudspeakers, about the tools and the materials one can use to manufacture loudspeakers;
- knowledge about the actions that may be performed to manufacture loudspeakers;
- knowledge about general technological solutions for the manufacturing of loudspeakers;
- knowledge to choose between various solutions of the problems to solve.

These types of knowledge are represented into the knowledge base of DISCIPLE in the form of object descriptions, action models, problem reduction rules, and meta-rules, respectively.

An object is described by specifying its relevant factual properties, as well as its relations with other objects, as illustrated in figure 2.2.

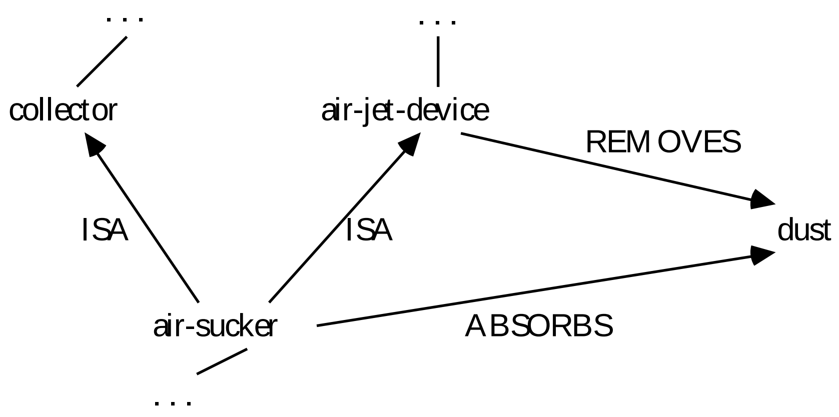


Figure 2.2. Object descriptions.

An action is described by specifying its name (as CLEAN in the example below), some of its cases (the object on which the action is performed, the instrument used, etc.), as well as the descriptions of these cases (which are always object descriptions):

CLEAN OBJECT entrefer
WITH air-sucker

Very important additional features of an action are its preconditions (i.e. the states of the world in which the action may be executed) and effects (i.e. the states that will result after the execution of the action). The traditional action planning systems make intensive use of these features. Nevertheless, when acquiring knowledge from an expert, requiring from him/her a complete description of the preconditions and effects of an action may quickly lead to a dead end in the relationship between system and human. This is why the preconditions and the effects are only optional features of an action. As we shall see, DISCIPLE is built precisely in order to overcome this problem.

The object descriptions and the action models are elementary knowledge about the application domain.

Using such elementary knowledge, DISCIPLE learns general problem reduction rules from examples of reductions provided by its user.

The meta-rules, for choosing between the rules applicable to reduce a problem, are supposed to be defined by the user, once such competing rules have been learned.

2.2.3 The problem solving method

Let us consider designing the manufacturing of some given loudspeaker.

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

MANUFACTURE OBJECT loudspeaker

DISCIPLE will try to solve this problem by successive decompositions and specializations, as illustrated in figure 2.3 and in figure 2.4.

In order to solve the problem
MANUFACTURE OBJECT loudspeaker
solve the subproblems
1. MAKE OBJECT chassis-assembly
   In order to solve this subproblem solve the sub-subproblems
   1.1 FIX OBJECTS 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 2.3. 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 the problem

CLEAN OBJECT entrefer WITH air-jet-device

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 2.4. Problem solving operations:
specializations of problems.

DISCIPLE will combine such decompositions and specializations building a problem solving tree like the one in figure 2.5.

This process continues until all the leaves of the tree are elementary actions, that is, actions which may be executed by the entity manufacturing the loudspeaker.

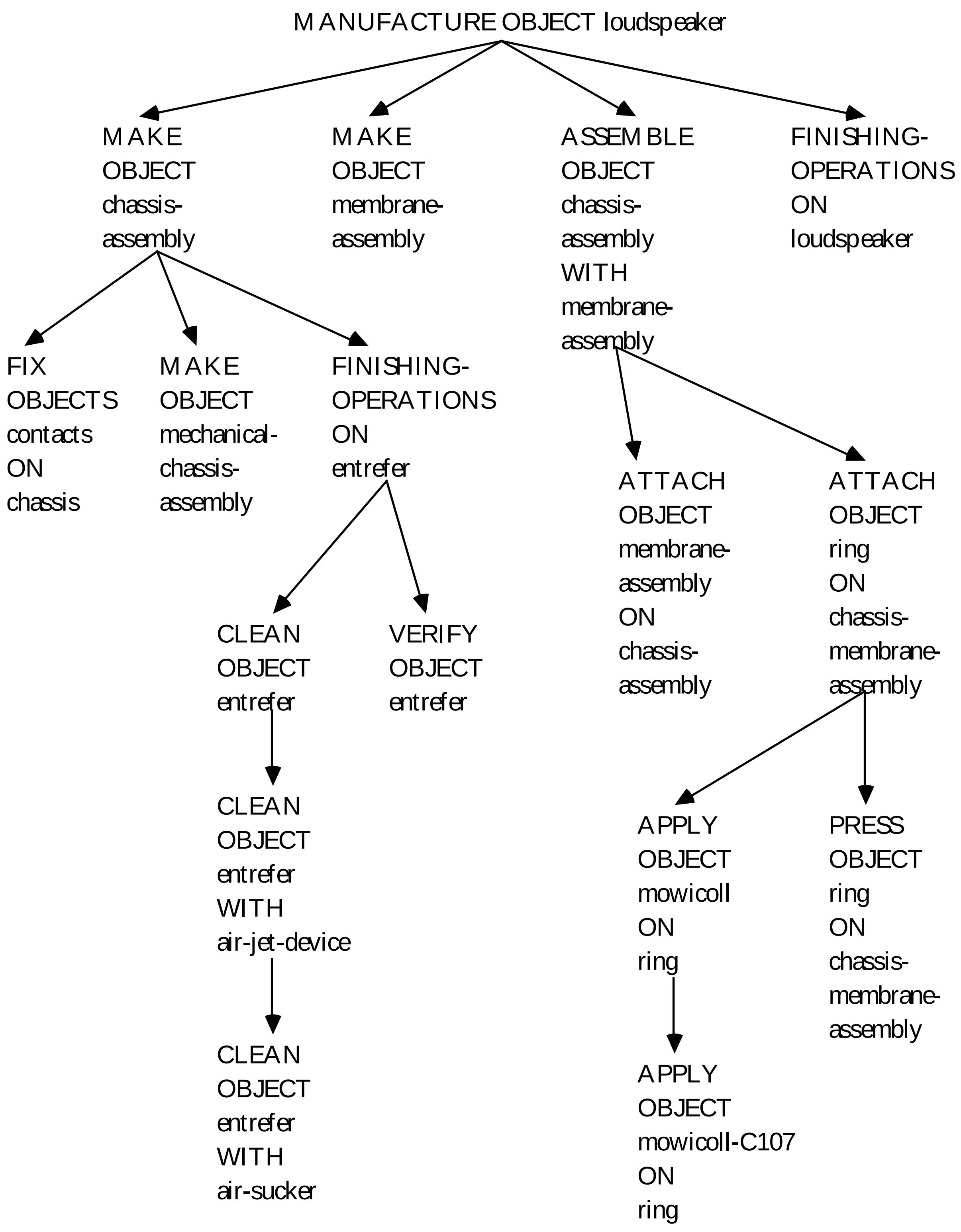

MANUFACTURE OBJECT loudspeaker
MAKE OBJECT chassis-assembly
MAKE OBJECT membrane-assembly
ASSEMBLE OBJECT chassis-assembly WITH membrane-assembly
FINISHING-OPERATIONS ON loudspeaker
FIX OBJECTS contacts ON chassis
MAKE OBJECT mechanical-chassis-assembly
FINISHING-OPERATIONS ON entrefer
ATTACH OBJECT membrane-assembly ON chassis-assembly
ATTACH OBJECT ring ON chassis-membrane-assembly
CLEAN OBJECT entrefer
VERIFY OBJECT entrefer
CLEAN OBJECT entrefer WITH air-jet-device
CLEAN OBJECT entrefer WITH air-sucker
APPLY OBJECT mowicoll ON ring
PRESS OBJECT ring ON chassis-membrane-assembly
APPLY OBJECT mowicoll-C107 ON ring

Figure 2.5. A problem solving tree.
The tree was built by using the decompositions from
figure 2.3 and the specializations from figure 2.4.

This is 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
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
```

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 and successively imposes different constraints on the specification and reduces object design to subparts design.

2.4 DISCIPLE as a Learning System

The reductions in figure 2.5 resulted from the application of reduction rules or were indicated by the user.

From each reduction (decomposition or specialization) indicated by the user, DISCIPLE will try to learn a general reduction rule.

Let us suppose, for instance, that DISCIPLE was not able to solve the problem:

ATTACH OBJECT ring ON chassis-membrane-assembly

and that the solution was indicated by the user:

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 2.6. A decomposition indicated by the user.

From this example, representing the decomposition of the problem of attaching two particular parts of the loudspeaker to a process of gluing, DISCIPLE learns a general decomposition rule indicating the conditions under which one may reduce an 'attachment' problem to a process of gluing:

IF
condition
(x TYPE solid) & (y TYPE solid) &
(x PARTIALLY-FITS y) &
(z ISA adhesive) & (z TYPE fluid) &
(z GLUES x) & (z GLUES y)
THEN
solve the problem
ATTACH OBJECT x ON y
by solving the subproblems
APPLY OBJECT z ON x
PRESS OBJECT x ON y

Figure 2.7. The general decomposition rule learned from Example 1: if 'x' and 'y' are two solid objects that partially fits 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 pressing 'x' on 'y'.

Once learned, this rule may be used to reduce other attachment problems to gluing processes. For instance, it may be used to reduce the problem

ATTACH OBJECT membrane-assembly ON chassis-assembly

from the problem solving tree in figure 2.5, to the following subproblems:

APPLY OBJECT neoprene ON membrane-assembly
PRESS OBJECT membrane-assembly ON chassis-assembly

As one may notice, the structure of the learned rule is identical with the structure of the example. 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' to be reduced to a process of gluing them with 'z'. That is, one should learn the concepts represented by these objects.

The method of learning this rule depends of the knowledge (theory) of the system about **Example 1**.

We have considered three types of theories: complete, weak, and incomplete.

A complete theory about Example 1 consists of the complete descriptions of all the objects and the actions of this example.

In this 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 from this example. It differs qualitatively from a complete theory in that it does not contain action models.

In this case, DISCIPLE uses an interactive learning method that synergistically combines explanation-based learning, learning by analogy, and empirical learning.

The intermediate case, between a complete theory and a weak theory, is the incomplete theory. As compared to a complete theory, an incomplete theory may lack some object descriptions or action models. Also, it may contain incomplete descriptions of objects and actions. As compared to a weak theory, an incomplete theory contains action models, while a weak theory does not.

In this case, DISCIPLE learns the decomposition rule in figure 2.7 by combining the method corresponding to the weak theory with the method corresponding to the complete theory.

Although, in each of the above cases, the system learns the same rule, this rule has a different impact on the future behavior of the system. In a complete theory, it improves the performance of the system, in a weak theory it develops the competence of the system, and, in an incomplete theory, it develops both the performance and the competence.

An important side effect of learning in the context of a weak theory or an incomplete theory is that of developing the domain theory.

3. KNOWLEDGE REPRESENTATION

One of our prime concern was to define a knowledge representation and organization suitable for both problem solving and learning.

The knowledge base of DISCIPLE is organized around the notion of 'concept', supporting the fundamental operations with the concepts:

- comparing the generality of concepts;
- generalizing concepts;
- particularizing concepts.

The problem solving and learning mechanisms of DISCIPLE are based on these elementary operations.

The knowledge base contains object descriptions, action models, reduction rules, and meta-rules.

The object descriptions and the action models are concepts. The reduction rules and the meta-rules are more complex entities which are built with concepts.

In this section we define the notion of concept and the basic operations with concepts. The reduction rules are defined in section 4 and the meta-rules are defined in section 5.

3.1 Concepts

Let 'U' be a universe of instances.

An instance may represent an object, an action, a goal or a state.

Let 'SP' be a set of predicates over the universe 'U'.

Each predicate 'P' from 'SP' splits the set 'U' into two subsets:

$$E = \{ e \in U \mid (P\ e) = \text{TRUE} \}$$
$$C = \{ c \in U \mid (P\ c) = \text{FALSE} \}$$

One says that the predicate 'P' asserts the concept representing the set of instances 'e' having the property

'(P e) = TRUE'.

An instance 'e' from 'E' is called an **example** or a **positive example** of the concept asserted by 'P' (for short, the concept 'P'), and an instance 'c' from 'C' is called a **counterexample** or a **negative example** of the concept 'P'.

Let 'adhesive' be the concept representing the set of objects which could be used for gluing other objects.

To state that 'mowicoll' is a positive example of 'adhesive' one may write:

(adhesive mowicoll) = TRUE

or

(mowicoll ISA adhesive)

To state that 'water' is a negative example of 'adhesive' one may write:

(adhesive water) = FALSE

or

NOT(water ISA adhesive)

3.2 Representation language

As in [Kodratoff & Ganascia, 1986], we define a representation language as follows:

Let 'V' be a countable set of **variables**.

Let FF = F0 U F1 U ... U Fn U ... be a **family of functions**, where 'Fn' is the set of functions of arity 'n' and 'F0' is the set of constants.

The set of **terms** on 'V' and 'FF' is defined by:

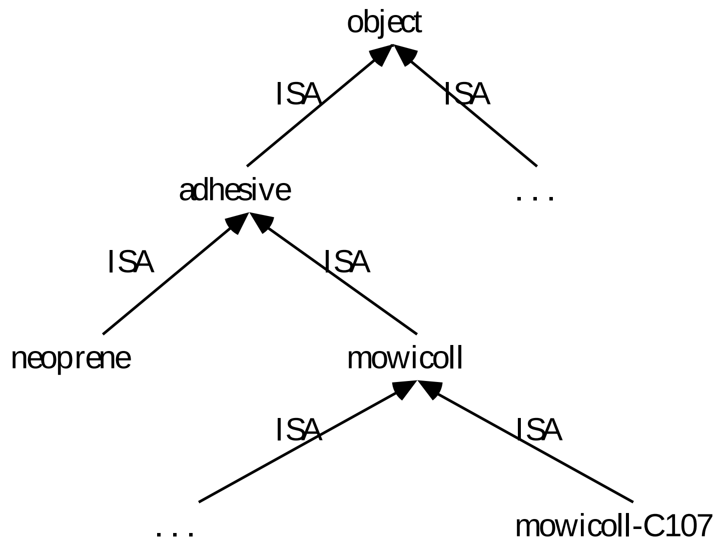
- v ∈ V is a term;

- f(t1,...,tn) is a term if and only if f□Fn and t1,...,tn are terms.

Thus, the set of terms is the set of expressions built with functions of some arity, constants, and variables.

Let SP = SP0 U SP1 U ... U SPn U ... be a set of **predicates**, where 'SPn' is the set of predicates of arity 'n' and 'SP0' is the set of the constants 'TRUE' and 'FALSE'.

These predicates may be related by one or several generalization hierarchies as, for instance, the following one:



Let 't1', ... , 'tn' be terms and 'p □ SPn'. Then '(p t1 ... tn)' and 'NOT(p t1 ... tn)' are called literals.

Because most of the used predicates are binary ones, we shall further write the literals in the following, more intuitive, form:

'(t1 p t2)'

The predicates may have properties or may be related by theorems as, for instance, the following ones:

commutativity:

∀x, ∀y (x NEAR y) —> (y NEAR x)

transitivity:
$\forall x, \forall y$ (x ON y) & (y ON z) ––> (x ON z)

theorem:
$\forall x, \forall y$ (x ABSORBS y) ––> (x GETS y)

The quadruple (FF, SP, H, L) is called a **representation language**, where H is the set of the theorems and properties of the predicates, variables, and constants, and L is the set of the logical connectors 'AND' and 'NOT', and of the connectors for specifying action concepts.

3.3 Object concepts

The knowledge base of DISCIPLE contains generic object concepts which are represented into hierarchical semantic networks, as the one in figure 3.1.

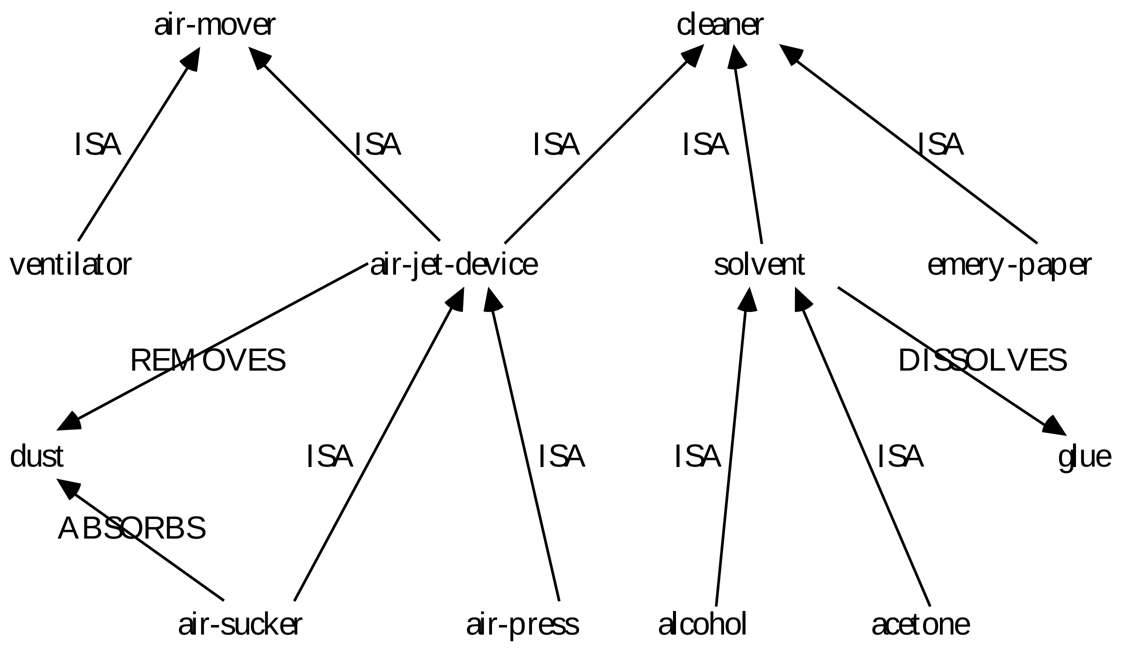


Figure 3.1. A hierarchical representation of generic object concepts.

In general, an object concept is defined as belonging to one or more super-concepts and having additional properties. The value of a property may be a constant or another concept.

Other object concepts may be defined in terms of such generic object concepts as, for instance, the following one, denoted by the variable 'y':

(y ISA cleaner) & (y REMOVES glue) &
NOT(y DESTROYS membrane)

Two types of theorems are implicitly represented into the concept hierarchies.

One is the transitivity of 'ISA' as, for instance,

(alcohol ISA solvent) & (solvent ISA cleaner)
—> (alcohol ISA cleaner)

or

∀x (x ISA ventilator) & (ventilator ISA air-mover)
—> (x ISA air-mover)

The other theorem which is implicitly represented into the object hierarchies is the inheritance of properties from a more general concept to a less general one as, for instance,

(air-jet-device REMOVES dust) & (air-press ISA air-jet-device)
—> (air-press REMOVES dust)

or

∀x (x ISA air-sucker) & (air-sucker ABSORBS dust)
—> (x ABSORBS dust)

3.4 Action concepts

In DISCIPLE, an action concept is defined by specifying its name, some of its cases [Filmore, 1968] (the agent performing the action, the object on which the action is performed, the instrument used, etc.), as well as the descriptions of these cases (which are always object descriptions).

For instance, the following action concept represents the set of all the cleaning actions where the object to be cleaned is an 'entrefer' and the instrument used is an 'air-sucker':

```
    CLEAN OBJECT e WITH a
    where (e ISA entrefer) & (a ISA air-sucker)
or
    CLEAN OBJECT (entrefer e)
            WITH   (air-sucker a)
```

Optional features of an action concept are its preconditions (i.e. the states of the world in which the action may be executed) and effects (i.e. the states of the world that will result after the execution of the action).

For instance, the following is a "complete" description of the action 'APPLY':

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

This action may be performed if and only if 'x' is a solid object and 'z' is a fluid object which is adherent on 'x'. As an effect of performing this action, 'z' will be applied on 'x'.

One may notice that the features of the objects are specified in the action's preconditions.

3.5 States and goals

A state of the world is a specification of all the objects and their current properties and relations.

A goal is a desired partial specification of a world state. That is, a goal is also a specification of objects.

For instance

(membrane-assembly ATTACHED-ON chassis-assembly)

represents the goal of having the membrane-assembly attached on the chassis-assembly. This goal is identified with the set of states in which this property holds.

3.6 Intuitive definition of generalization

Generalization is a fundamental property of a concept.

Intuitively, a concept 'P' is said to be **more general than** another concept 'Q' if 'P' takes the value TRUE in all the cases in which 'Q' takes the value TRUE.

Let 'P' and 'Q' be the concepts representing the following sets of instances:

{Ptrue} = { x | (P x) = TRUE }

{Qtrue} = { x | (Q x) = TRUE }

One says that the concept 'P' is **more general than** the concept 'Q' if and only if {Qtrue} is included into {Ptrue}.

Let 'mowicoll' be the set of adhesives of type mowicoll. The concept 'adhesive' is clearly more general than the concept 'mowicoll'. One may express it as follows:

(mowicoll ISA adhesive)

Notice that this definition of generalization is extensional, based upon the instance sets of concepts. It is useful in practice only for showing that 'Q' is not more general than 'P'. Indeed, in such a situation, it is enough to find an instance 'x' such that

(P x) = TRUE and (Q x) = FALSE

In order to show that 'P' is more general than 'Q', this definition would require the computation of the (possible infinite) sets of the instances of 'P' and 'Q'.

However, in a representation language, one may define a more operational definition of the **more general than** relation, as will be shown in the next section.

3.7 Generalization in a representation language

The generalization is defined in terms of substitutions, as in [Kodratoff & Ganascia, 1986].

A substitution has the following form:

σ = (x1<–f1, ... , xn<–fn)

In DISCIPLE, each 'xi' (i=1,...,n) is a variable or a concept and each 'fi' (i=1,...,n) is a term or a concept.

If 'xi' is a variable then 'fi' is a term, and if 'xi' is a concept then 'fi' is a less general concept (according to a generalization hierarchy from the representation language).

If 'li' is a literal, then 'σli' is the literal obtained by substituting each 'xi' from 'li' with 'fi'.

3.7.1 Term generalization

We say that the term 't1' is **more general than** the term 't2' if there exists a substitution σ such that σt1=t2.

Let t1 = VOLUME-CYLINDER(r h)

t2 = VOLUME-CYLINDER(3 7)

σ = (r<–3, h<–7)

σt1 = t2

so 't1' is more general than 't2'

This definition does not take into account the properties of the functions. In general, however, one needs to use these properties to show that two terms are equal.

Let t1 = VOLUME-CUBOID(x y z)
t2 = VOLUME-CUBOID(x z y)

To show that these terms are equal one needs to use the axiom of commutativity of the arguments of the function VOLUME-CUBOID.

Therefore, we say that the term 't1' is **more general than** the term 't2' iff there exist 't1"', 't2"', and a substitution σ, such that:

t1" = t1
t2" = t2
σt1"= t2"

Let
t1 = VOLUME-CUBOID(x y x)
t2 = VOLUME-CUBOID(5 5 7)
t1"= VOLUME-CUBOID(x x y)
σ = (x<–5, y<–7)
σt1" = t2.
Therefore, 't1' is more general than 't2'.

3.7.2 Literal generalization

The literal 'l1 = (p1 t11 ... t1n)' is **more general than** the literal 'l2 = (p2 t21 ... t2n)' if and only if there exist 'l1"', 'l2"', and 'σ' such that:

l1" = l1
l2" = l2
σl1"= l2"

If 'l1' is more general than 'l2' then 'p1 = p2' or 'p1' is more general than 'p2', according to a generalization hierarchy from the representation language.

3.7.3 Conjunctive formula generalization

Let us consider two conjunctive formulas:

$$A = A1 \,\&\, A2 \,\&\, \dots \,\&\, An$$
$$B = B1 \,\&\, B2 \,\&\, \dots \,\&\, Bm$$

where 'Ai' (i = 1, ... ,n) and 'Bj' (j = 1, ... ,m) are literals.

'A' is **more general than** 'B' if and only if there exist 'A"', 'B"', and 'σ' such that:

$A'' = A, \quad A'' = A1'' \,\&\, A2'' \,\&\, \dots \,\&\, Ap''$

$B'' = B, \quad B'' = B1'' \,\&\, B2'' \,\&\, \dots \,\&\, Bq''$

$\forall\, i \in \{1,\dots,p\},\ \exists\, j \in \{1,\dots,q\}$ such that $\sigma Ai'' = Bj''$.

Otherwise stated, one transforms the formulas 'A' and 'B', using the theorems and the properties of the representation language, so that to make each literal from 'A"' more general than a corresponding literal from 'B"'.

Notice that some literals from 'B"' may be "left-over", i.e. they are matched by no literal of 'A"', as in the following example.

Let

A = (u ISA adhesive) & (u GLUES v) & (u GLUES w)

B = (x ISA adhesive) & (x GLUES y) & (x TYPE fluid)

One may transform 'B' by using the idempotence of the '&' operator (P=P&P), thus obtaining:

B"= (x ISA adhesive) & (x GLUES y) & (x GLUES y) &
(x TYPE fluid)

There exists 'σ = (u<–x, v<–y, w<–y)' such that:

σA = (x ISA adhesive) & (x GLUES y) & (x GLUES y)

That is, 'σA' is a part of 'B"'. Therefore, 'A' is more general than 'B'.

We have defined the more general relation in the context of the representation language of DISCIPLE.

In the next section we shall show how one can effectively generalize or particularize concepts, by applying syntactic rules of generalizations/particularizations (which may be regarded as elementary substitutions).

3.8 Syntactic rules of generalization/particularization

To generalize concepts, DISCIPLE uses syntactic rules of generalization [Michalski, 1983; Kodratoff & Ganascia, 1986].

It is important to notice that these rules do not preserve the truth. That is, if 'A' is true and is generalized to 'B', then the truth of 'B' is not guaranteed. Therefore, one fundamental problem of the learning systems is to make those generalizations that preserve the truth.

To particularize concepts, one may use the same rules in the reverse order.

3.8.1 Turning constants into variables

This rule consists in generalizing an expression by replacing a constant with a variable.

For instance, the expression

(x ISA cleaner) & (x ABSORBS dust)

may be generalized to

(x ISA cleaner) & (x ABSORBS y)

by turning the constant 'dust' into the variable 'y'.

The first expression represents the set of 'cleaners' that absorb 'dust' while the second expression represents the set of 'cleaners' that absorbs something. Since the second set includes the first one, it is more general.

3.8.2 Turning occurrences of a variable into different variables

According to this rule, the expression

```
TIN OBJECTS (terminal-wires t)
    WITH    (tinning-bath x)
TIN OBJECTS (coil-ends c)
    WITH    (tinning-bath x)
```

may be generalized to

```
TIN OBJECTS (terminal-wires t)
     WITH    (tinning-bath u)
TIN OBJECTS (coil-ends c)
     WITH    (tinning-bath v)
```

The first expression represents the set of the sequences consisting of two TIN operations, both using the same 'tinning-bath' named 'x'.

The second expression represents the set of the sequences consisting of two TIN operations, using the 'tinning-bathes' named 'u' and 'v', respectively.

In particular, 'u' and 'v' may represent the same 'tinning-bath'. Therefore, the second set includes the first one, and the second expression is more general than the first one.

3.8.3 Climbing the generalization hierarchies

The concepts from DISCIPLE's knowledge base are organized into generalization hierarchies, as has been shown in section 3.3.

Using such hierarchies, DISCIPLE may generalize an expression by replacing a concept with a more general one.

For instance, the expression

(x ISA emery-paper) & (x REMOVES y)

may be generalized to

(x ISA cleaner) & (x REMOVES y)

by replacing the concept 'emery-paper' with the more general concept 'cleaner'.

3.8.4 Dropping conditions

This rule consists in generalizing a concept by removing a constraint from its description [Kodratoff & al. 1984].

For instance, the expression (x ISA adhesive) & (x TYPE fluid)
may be generalized to (x ISA adhesive)
by removing a constraint on the 'adhesive' to be of type fluid.

3.8.5 Using theorems

The knowledge base of DISCIPLE also contains theorems for deducing properties and relations between concepts from other properties and relations. If 'B–>C' is such a theorem, then one may generalize 'B' to 'C', or 'A & B' to 'A & C'.

For instance, using the theorem
∀x ∀y [(x GLUED-ON y) –> (x ATTACHED-ON y)]
one may generalize the expression
(r ISA ring) & (c ISA chassis-membrane-assembly) &
(r GLUED-ON c)
to
(r ISA ring) & (c ISA chassis-membrane-assembly) &
(r ATTACHED-ON c)

As has be shown in section 3.3, two types of theorems are implicitly represented into the concept hierarchies. They are the transitivity of the generalization relation and the inheritance of properties from a more general concept to a less general one.

Let us notice that the above syntactic rules of generalization are, in fact, special cases of using theorems rule [Vrain, 1987]. However, making them explicit, improves the efficiency of the system.

3.8.6 Structural generalization

This rule consists in the reverse application of a decomposition rule 'A ––> C(A1, ... ,An) '.

Let us consider, for instance, the following decomposition rule:

```
IF
    NOT (x MATERIAL fragile) &
    (x THICKNESS thick) &
    (y MATERIAL rigid)
THEN
solve the problem
    ATTACH OBJECT x ON y
by solving the subproblems
    MAKE OBJECTS rivets ON x
    MAKE OBJECTS holes IN y
    RIVET OBJECT x WITH y
```

This rule suggests that we might generalize the following sequence of actions

```
    MAKE OBJECTS rivets ON (upper-flange u)
    MAKE OBJECTS holes IN (chassis c)
    RIVET OBJECT u WITH c
to
    ATTACH OBJECT (upper-flange u) ON (chassis c)
```

The second expression represents the set of the technological solutions for attaching an 'upper-flange' on a 'chassis'. This set includes, among others, the attachment by riveting (that is, the technological solution represented by the first expression).

4. PROBLEM SOLVING MECHANISMS

The problem solving mechanisms of DISCIPLE are based on techniques of problem reduction [Nilsson, 1971], formulation, propagation and evaluation of constraints [Stefik, 1980], and problem solving by analogy [Carbonell, 1986]. All these techniques are reduced to the basic operations with concepts, and are integrated into a unitary problem reduction method.

In the following we shall describe these problem solving mechanisms.

4.1 Problem reduction

4.1.1 The problem reduction method

The problem reduction method may be formulated as follows:

Given:
- the description of an initial problem;
- a set of reduction rules for transforming problems into subproblems;
- a set of primitive problems, i.e. problems with known solutions.

Determine:
- a set of primitive problems that solve the initial problem, by successively applying the reduction rules.

Figure 4.1. The problem reduction method.

4.1.2 Decomposition rules

In DISCIPLE, a reduction rule has the following form:

```
IF
        condition
        K(x1,...,xn,...,xq)
THEN
solve the problem
        P(x1,...,xn)
by solving the subproblems
        C( P1(x1,...,xn,...,xq),
         P2(x1,...,xn,...,xq),
          ...
         Pm(x1,...,xn,...,xq) )
```

This rule indicates the decomposition of the problem 'P' into a set of subproblems 'P1', 'P2', ... , 'Pm'. Therefore we shall call such a rule a **decomposition rule**.

'C' is a combinator which indicates the way of combining the solutions of the problems 'P1', 'P2', ... , 'Pm', in order to obtain the solution of the problem 'P'.

'K' is a conjunction of predicates that have to be true in order for the rule to be applicable.

One should notice that 'K', 'P1', 'P2',..., and 'Pm' may contain variables not present in 'P'. These variables are supposed to be existentially quantified.

The rule form given here is general in that it does not represent the decomposition of a specific problem, but the decomposition of an entire set of problems. This set contains any problem which is less general than 'P' and satisfies 'K'. It is defined as follows:

$\{ Pa \mid \text{there exists } \sigma \text{ such that } \sigma P = Pa \text{ and } \sigma K = \text{TRUE} \}$

Let 'Pa' be the problem to be solved and 'σ' a substitution such that 'σP = Pa' and 'σK = TRUE'.

Then, the above rule indicates the following decomposition of the problem 'Pa': 'C(σP1, σP2, ... , σPm)'.

4.1.3 Decomposition of problems

Let us consider the following decomposition rule, indicating a way to perform the attachment of two objects:

```
IF
    (x TYPE solid) &
    (y TYPE solid) &
    (x PARTIALLY-FITS y) &
    (z ISA adhesive) &
    (z TYPE fluid) &
    (z GLUES x) &
    (z GLUES y)
THEN
solve the problem
    ATTACH OBJECT x ON y
by solving the subproblems
    APPLY OBJECT z ON x
    PRESS OBJECT x ON y
```

The variable 'z', in the condition of this rule, is considered to be existentially quantified.

Let us now consider the following problem to solve:

ATTACH OBJECT membrane-assembly ON chassis-assembly

In order to decompose this problem one must first find a substitution 'σ' such that:

σ(ATTACH OBJECT x ON y) =
ATTACH OBJECT membrane-assembly ON chassis-assembly

This substitution is

σ = (x <– membrane-assembly, y <– chassis-assembly)

Next, one has to verify that, in the current situation,

σ(condition) = TRUE

That is, one has to verify that the following expression

(membrane-assembly TYPE solid) &
(chassis-assembly TYPE solid) &
(membrane-assembly PARTIALLY-FITS chassis-assembly)&
(z ISA adhesive) &
(z TYPE fluid) &
(z GLUES membrane-assembly) &
(z GLUES chassis-assembly)

is true, where the variable 'z' is considered to be existentially quantified.

If the above expression is true, then one can solve the problem

ATTACH OBJECT membrane-assembly ON chassis-assembly

by solving the subproblems

APPLY OBJECT z ON membrane-assembly
PRESS OBJECT membrane-assembly ON chassis-assembly
where: (z ISA adhesive) &
                (z GLUES membrane-assembly) &
                (z GLUES chassis-assembly) &
                (z TYPE fluid)

Notice that the choice of an appropriate adhesive is a problem to be addressed later.

Another decomposition rule is the following one:

```
IF
        (l ISA loudspeaker) &
        (l HAS s) &
        (s ISA screening-cap)
THEN
solve the problem
        FINISHING-OPERATIONS  ON  l
by solving the subproblems
        VERIFY  OBJECT   l
        ATTACH  OBJECT  s  ON  l
        MARK  OBJECT  l
```

This rule states that the finishing operations for a 'loudspeaker' having a 'screening-cap' consist in verifying the 'loudspeaker', attaching the 'screening-cap' on the 'loudspeaker', and marking the 'loudspeaker'.

4.2 Problem solving by constraints

DISCIPLE is designed to solve problems which are initially imprecisely formulated but become better and better formulated as the problem solving process advances. To this purpose it formulates, propagates, and evaluates constraints.

4.2.1 Constraint formulation

Constraint formulation is the process of imposing constraints on the problems to solve.

One way of formulating constraints in DISCIPLE is to apply specialization rules.

A specialization rule has the following form:

```
IF
        condition
        K(x1,...,xn,...,xq)
THEN
solve the problem
        P(x1,...,xn)
by solving the specialization
        Pi(x1,...,xn,...,xq)
```

This rule states that if the condition 'K' is satisfied then the problem 'P' may be constrained to the problem 'Pi'.

In DISCIPLE, constraining a problem means providing a more precise specification of it.

As in the case of the decomposition rules, a specialization rule is general in that 'P' does not represent a particular problem, but an entire set of problems. This set contains any problem that is less general than 'P' and satisfies 'K'. It is defined as follows:

$$\{ Pa \mid \text{there exists } \sigma \text{ so that } \sigma P = Pa \text{ and } \sigma K = TRUE \}$$

Given a problem 'Pa' from this set, the above rule will suggest to constrain 'Pa' to 'σPi'.

If 'P' represents an action, then a specialization rule may constrain the action to use a certain instrument, as in the case of the following examples:

```
IF
    (x HAS z) &
    (z ISA waste-material) &
    (y ISA cleaner) &
    (y REMOVES z) &
    NOT(y DESTROYS x)
THEN
solve the problem
    CLEAN OBJECT x
by solving the specialization
    CLEAN OBJECT x WITH y
```

Figure 4.2. A specialization rule.
This rule states that, in order to clean an object 'x' which presents a waste material 'z', one should use a cleaner 'y' which removes 'z' but does not destroys 'x'.

Another specialization rule is the following one:

```
IF
    (y ISA feature) &
    (z ISA measurement-instrument) &
    (z MEASURES y)
THEN
solve the problem
    VERIFY OBJECT x
        FEATURE y
by solving the specialization
    VERIFY  OBJECT x
        FEATURE y
        INSTRUMENT z
```

Figure 4.3. Another specialization rule.
This rule states that, in order to verify a feature 'y' of an object 'x', one should use a measurement instrument 'z' which measures 'y'.

If the current problem is to design an object, then a specialization rule may specify a new feature of the object.

DISCIPLE solves problems in a hierarchical manner, formulating more and more detailed constraints as the problem solving process advances. In this way, it is not forced to consider all the details from the very beginning, but only when they are needed.

4.2.2 Constraint propagation

Constraint propagation is the process of transmitting the constraints from one problem to another problem. This is the way the problems communicate between themselves.

A problem is usually under-constrained in that there are many possible specializations of it which achieve the desired goal.

By constraint propagation this problem receives constraints from other problems.

Constraint propagation thus makes a **least-commitment strategy** possible. That is, DISCIPLE keeps open its options and reasons by elimination when constraints from other problems are received.

In DISCIPLE, constraint propagation is an instantaneous process due to the fact that the objects from the problems to solve are uniquely represented.

Let us consider, for instance, two tinning operations, each using the same tinning-bath 'x':

```
TIN OBJECTS (terminal-wires t)
    WITH    (tinning-bath x)
TIN OBJECTS (coil-ends c)
    WITH    (tinning-bath x)
```

DISCIPLE maintains a unique description of 'x'. Therefore, if one of these two operations is specialized by imposing a constraint to the tinning-bath 'x', then the other tinning operation is automatically specialized, since it refers the same object 'x'.

DISCIPLE maintains a history of the successive constraints of a variable, being able to undo these constraints when it fails to solve a problem.

4.2.3 Constraint evaluation

Constraint evaluation is the process of determining values for variables, values satisfying the constraints imposed on the variables.

In DISCIPLE, the constraints describe the features which are needed for the objects specified in the problems to be solved.

Constraint evaluation is a decision problem of the type **buy or build** [Stefik, 1980]. First of all, DISCIPLE looks in its knowledge base for an object that satisfies the constraints. If no object is found then the system will try to design the object. Thus, the design of the object becomes a new problem to solve.

Let us consider, for instance, the following description:

(c ISA cleaner) & (c REMOVES t)

and the object hierarchy in figure 3.1.

Each object concept from the hierarchy in figure 3.1 which is less general than 'c' represents a valid value for 'c'.

Following its least commitment strategy, DISCIPLE will choose the most general possible concept among the ones less general than 'c'. That is, the above concept may be replaced with 'air-jet-device' (by turning the variable 't' into the constant 'dust') or with 'solvent' (by turning 't' into 'glue').

The specialization rules proved very useful in the technology design domain, allowing to introduce into an action description information concerning tools, devices, verifiers or materials.

4.3. Problem solving by analogy

[Carbonell, 1986] defines problem solving by analogy as the process of "transferring knowledge from past problem-solving episodes to new problems that share significant aspects with corresponding past experience and using the transferred knowledge to construct solutions to the new problems."

Two problems are considered to share significant aspects if they match within a certain threshold, according to a given similarity metric. In this case, the past problem solving episode is perturbed in such a way as to satisfy the requirements of the new problem.

From the past problem solving episodes DISCIPLE learns rules. Therefore, a rule represents a generalized problem solving episode. However, if the domain theory is incomplete or weak, the learned rules may not specify a single applicability condition (as was presented in the previous sections) but two conditions (as will be presented in sections 8 and 9):

```
IF
        analogy criterion
        Kg(x1,...,xn,...,xq)
        condition
        K(x1,...,xn,...,xq)
THEN
solve the problem
        P(x1,...,xn)
by solving the subproblems
        C( P1(x1,...,xn,...,xq),
         P2(x1,...,xn,...,xq),
          ...
         Pm(x1,...,xn,...,xq) )
```

If the problem to solve does not satisfy the condition of the rule but does satisfy the *analogy criterion*, then one may look for a reduction of the problem by analogy with the reduction specified in the rule.

Let 'Pa' be the current problem to solve.

If there exists a substitution 'σ' such that 'σP = Pa' and 'σK = TRUE' then 'C(σP1, σP2, ... , σPm)' is a decomposition of 'Pa', no analogy process taking place.

If 'σK = FALSE' but 'σKg = TRUE' then 'C(σP1, σP2, ... ,σPm)' is a decomposition of 'Pa', obtained by analogy with the decomposition of 'P' to 'C(P1, P2, ... , Pm)'.

We do not go into further details on this process since it will become evident after the presentation of the learning methods of DISCIPLE.

5. CONTROL MECHANISMS

5.1 Definition of the search space

Let us suppose that DISCIPLE has to solve the problem 'Pa'. In principle, there may be several applicable rules, each indicating a partial solution to 'Pa' (a decomposition or a specialization).

Let R1, R2, and R3 be the applicable rules which suggest the reduction of 'Pa' to 'C(Pb,Pc)', 'Pd', and 'C(Pe,Pf,Pg)', respectively.

Therefore, to solve the problem 'Pa', one may either:
- solve the problems 'Pb' and 'Pc', or
- solve the problem 'Pd', or
- solve the problems 'Pe', 'Pf' and 'Pg'.

One may represent all these alternatives in the form of an AND/OR tree:

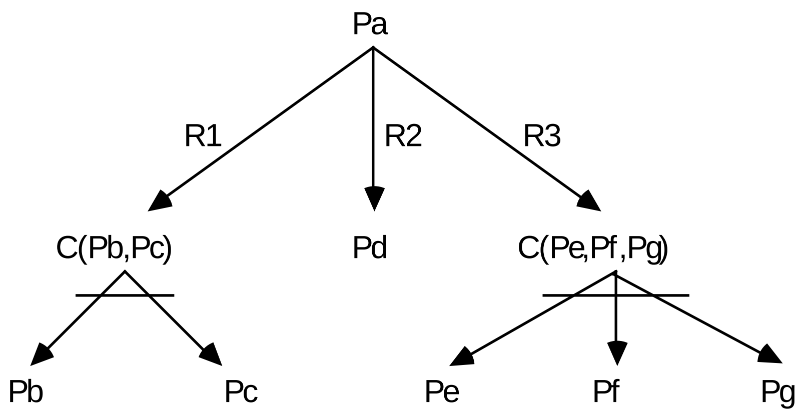


Figure 5.1 An AND/OR tree.

The node 'Pa' is called an **OR** node since for solving the problem 'Pa' it is enough to solve 'C(Pb, Pc)' **OR** 'Pd' **OR** 'C(Pe, Pf, Pg)'.

The node 'C(Pb, Pc)' is called an **AND** node since for solving it one must solve 'Pb' **AND** 'Pc'.

This tree may still be developed by considering all the rules applicable to its leaves (Pb, Pc, Pd, Pe, Pf, Pg). In this way one may build the entire **search space** for the problem 'Pa'. This space contains all the solutions to 'Pa'.

Usually, however, one solution is enough. To find it one needs only to build enough of the tree to demonstrate that 'Pa' is solved. Such a tree is called a **solution tree**.

A node is **solved** in one of the following cases:

1. it is a terminal node (a primitive problem);
2. it is an AND node whose successors are solved;
3. it is an OR node which has at least one solved successor.

Similarly, a node is **unsolvable** in one of the following cases:

1. it is a nonterminal node that has no successors (a nonprimitive problem to which no rule applies)
2. it is an AND node which has at least one unsolvable successor.
3. it is an OR node which has all the successors unsolvable.

To solve the problem 'Pa', one has to build a solution tree. Once the problem solver detects that a node is solved or unsolvable it sends this information to the ancestors of the node. When the node 'Pa' becomes solved, one has found a solution to 'Pa'. If the node 'Pa' becomes unsolvable, then no solution to 'Pa' exists.

The presented method supposes an exhaustive search of the solution space. Usually, however, the real world problems are characterized by huge search spaces and one has to use heuristic methods in order to limit the search.

[Feigenbaum & Feldman, 1963] defines the heuristic as being "a rule of thumb, strategy, trick, simplification, or any other kind of device which drastically limits search for solutions in large problem spaces. Heuristics do not guarantee optimal solutions; in fact they do not guarantee any solution at all; all that can be said for a useful heuristic is that it offers solutions which are good enough most of the time".

One may distinguish between two types of control decisions:

1. **Attention focusing**: what problem, among the leaves of the problem solving tree, to reduce next ?

2. **Meta-rule**: what rule, among the applicable ones, to use for reducing the current problem ?

The first type of decision establishes the global strategy for searching the solution tree.

One may distinguish between different global search strategies:

- breadth first search;
- depth first search;
- heuristic search (the heuristics establish the next problem to solve);
- user directed search (the user establishes the next problem to solve);
- etc.

The second type of decision consists in choosing the solution of a problem, in order to optimize certain criteria.

In the next sections we shall discuss these two types of decisions in detail.

5.2 Global control of the search

The AND/OR tree in figure 5.1 supposes parallel development of several solution paths.

In the current implementation of DISCIPLE one develops an AND tree as the one in figure 2.5 and backtracks when problem solving fails.

Otherwise stated, DISCIPLE will not develop the problem solving tree in figure 5.1. To reduce the problem 'Pa' it will apply only one of the rules R1, R2 or R3 as, for instance, R1. When it will discover that this path does not lead to a solution, it will backtrack to 'Pa' and will reduce it with another rule.

The global control in DISCIPLE is based on the agenda mechanism [Stefik, 1980].

DISCIPLE maintains an agenda of the problems to be solved. Initially this agenda contains only the problem indicated by the user.

This initial problem evolves in a problem solving tree, as it is successively decomposed and specialized (see figure 2.5).

Each leaf of this tree is a new problem to reduce.

First, one has to decide on the next problem to reduce, among the leaf problems.

Next, one has to establish the kind of reduction to apply:

- to decompose the problem to a set of subproblems;
- to specialize the problem to a better defined one;
- to specialize an object from the problem.

Although such a control strategy might itself be learned, we have not yet considered this learning task. Instead, several control strategies (depending on the application domain) are easy to implement into the system. Such a special control strategy will be presented in section 10.1.

Depending on the implemented control strategy, the system may decide itself which problem to reduce next and what kind of rule to look for. However, for real world applications, it is important that the user himself be able to control the problem solving process. To this purpose, he may use the following commands:

- decompose the current problem;
- specialize the current problem to a better defined one;
- specialize an object from the current problem;
- print the current partial solution (the leaves of the current problem solving tree);
- delete the subtree of the current problem (that is, disregard the partial solution of the problem);

as well as commands for crossing the current problem-solving tree: move to the father (first son, left brother, left-most brother, right brother, right-most brother, etc) of the current problem

5.3. Meta-rules

Once DISCIPLE has decided on the type of rule to apply, it has to choose among the applicable rules. The Expert Systems literature calls this the **conflict resolution problem**.

This type of decision is modeled in DISCIPLE with so-called **meta-rules**.

The meta-rules are heuristics for ordering the rules applicable to reduce a problem 'P', in order to optimize certain criteria. They have the following form:

To solve the problem
P
optimizing the criterion
O C
consider the applicable rules in the following order
R1, ... , Rn

This meta-rule defines an order on the rules that may reduce the problem 'P' in that 'R1' is expected to give the best result and 'Rn' the worst, with respect to the optimization criterion OC.

Of course, other criteria may require other orderings on the same set of rules.

A heuristic search consists in using only those rules which are recommended by meta-rules ('R1' in our case) and not all the applicable rules (R1,R2, ... ,Rn).

Only if 'R1' does not lead to a solution, the system will use 'R2'.

The following is an example of meta-rule:

```
To solve the problem
ATTACH OBJECT x ON y
optimizing the criterion
INCREASE shock-resistance
consider the applicable rules in the following order
1.
IF
    NOT (x MATERIAL fragile) &
    (x THICKNESS thick) &
    (y MATERIAL rigid)
  THEN
  solve the above problem by solving the subproblems
    MAKE OBJECTS rivets ON x
    MAKE OBJECTS holes IN y
    RIVET OBJECT x WITH y
2.
IF
    (x TYPE solid) &
    (y TYPE solid) &
    (x PARTIALLY-FITS y) &
    (z ISA adhesive) &
    (z TYPE fluid) &
    (z GLUES x) &
    (z GLUES y)
  THEN
  solve the above problem by solving the subproblems
    APPLY OBJECT z ON x
    PRESS OBJECT x ON y
```

This meta-rule states that the attachment by riveting has a better shock-resistance than the attachment by gluing.

It is interesting to notice that the meta-rules corresponding to object specializations may be associated with the nodes in the object hierarchies.

Let us consider, for instance, the object hierarchy in figure 3.1.

An object concept from such a hierarchy may be specialized to any of its sons. Therefore, the conflict set associated with an object concept consists of all the sons of this concept.

For instance, the conflict set associated with the concept

'air-jet-device' is {air-sucker, air-press}

since each element of this set may specialize the 'air jet device' concept, in a problem description containing it.

A meta-rule associated with the object 'air-jet-device', in the hierarchy from figure 3.1, may be the following one:

To specialize the concept
air-jet-device
optimizing the criterion
INCREASE work-safety
consider the specializations in the following order
1. air-sucker
2. air-press

That is, if we want to increase the work-safety, then it is better to specialize 'air-jet-device' to 'air-sucker' than to 'air-press'.

The following meta-rules show that different optimizing criteria may impose different orderings on the same conflict set:

To specialize the concept
dryer
optimizing the criterion
DECREASE time
consider the specializations in the following order
1. tunnel-kiln
2. carrousel
3. drying-shelf

To specialize the concept
    dryer
optimizing the criterion
    DECREASE cost
consider the specializations in the following order
    1. drying-shelf
    2. carrousel
    3. tunnel-kiln

To choose between such competing meta-rules, the system will ask the user to specify the optimization criterion.

6. THE LEARNING PROBLEM

DISCIPLE is trying to learn a general problem solving rule from each example received from the user.

The learning problem of DISCIPLE may be formulated as follows:

Given:
- a domain theory;
- a problem to solve and a partial solution to the problem, in the form of a decomposition or a specialization of it.

Determine:
- a general decomposition or specialization rule.

Figure 6.1 The learning problem of DISCIPLE.

For instance,

Given:

- the theory of loudspeaker manufacturing;

- the problem of attaching two parts of the loudspeaker and the decomposition of this problem into two simpler subproblems expressing the gluing of the two parts with an adhesive:

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 6.2. A decomposition indicated by the user.

Determine:

- a general decomposition rule indicating the conditions under which one may reduce an 'attachment' problem to a process of gluing:

```
IF
    x, y, and z satisfy <constraints>
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 6.3. The decomposition rule to be learned from Example 1

As one may notice, the structure of **General Rule 1** is identical with the structure of **Example 1**. 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 'ring', 'chassis-membrane-assembly', 'mowicoll', and of the actions 'ATTACH', 'APPLY', 'PRESS'.

A complete description of an object specifies all the relevant factual properties of the object, as well as all the relevant relations with other objects. These may be explicitly specified or may be deduced by using inference rules.

A complete description of an action is an action model that specifies all the necessary preconditions of the action, all its effects, as well as, all the objects that may play certain roles in the action.

In the case of a complete theory, 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 'ring', 'chassis-membrane-assembly', and 'mowicoll'. It differs qualitatively from a complete theory in that it does not contain the models of the actions 'ATTACH', 'APPLY', and 'PRESS'.

In this case, DISCIPLE uses an interactive learning method that synergistically combines explanation-based learning, learning by analogy, and empirical learning.

The intermediate case, between a complete theory and a weak theory, is the **incomplete theory**.

As compared to a complete theory, an incomplete theory may lack some object descriptions or action models. Also, it may contain incomplete descriptions of objects and actions.

As compared to a weak theory, an incomplete theory contains action models while a weak theory does not.

An incomplete description of an object lacks some important properties or relations of this object. Also, some inference rules for deducing new properties and relations of this object may be incompletely specified.

An incomplete description of an action lacks some precondition or effect predicates.

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.

It is interesting to notice that the effect of the learned rule on the future behaviour of the system depends on the type 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 *i**ncomplete theory***, it develops both the ***performance*** and the ***competence*.**

Another goal of learning in the context of a weak or incomplete theory is that of developing the domain theory.

The reason for dealing with all these three types of theories in a single system is the following one. ***Most domain theories are nonhomogeneous*** in that they provide complete descriptions of some parts of the domain, and incomplete or even weak descriptions of other parts of the domain. A learning episode, however, uses only one part of the domain theory that may have the features of a complete, incomplete or weak theory, even if, globally, the theory is nonhomogeneous. Therefore, the learning system has to adopt the learning method corresponding to the features of the theory about the example from which it learns.

In the following sections we shall present these three learning methods of DISCIPLE.

7. LEARNING IN A COMPLETE THEORY DOMAIN

7.1 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 6.2, contains the complete descriptions of the objects 'ring', 'chassis-membrane-assembly', and 'mowicoll', as well as the complete models of the actions 'ATTACH', 'APPLY', and 'PRESS'.

The objects are described by specifying all the relevant factual properties and relations.

Some object properties and relations may be explicitly specified, as indicated in figure 7.1

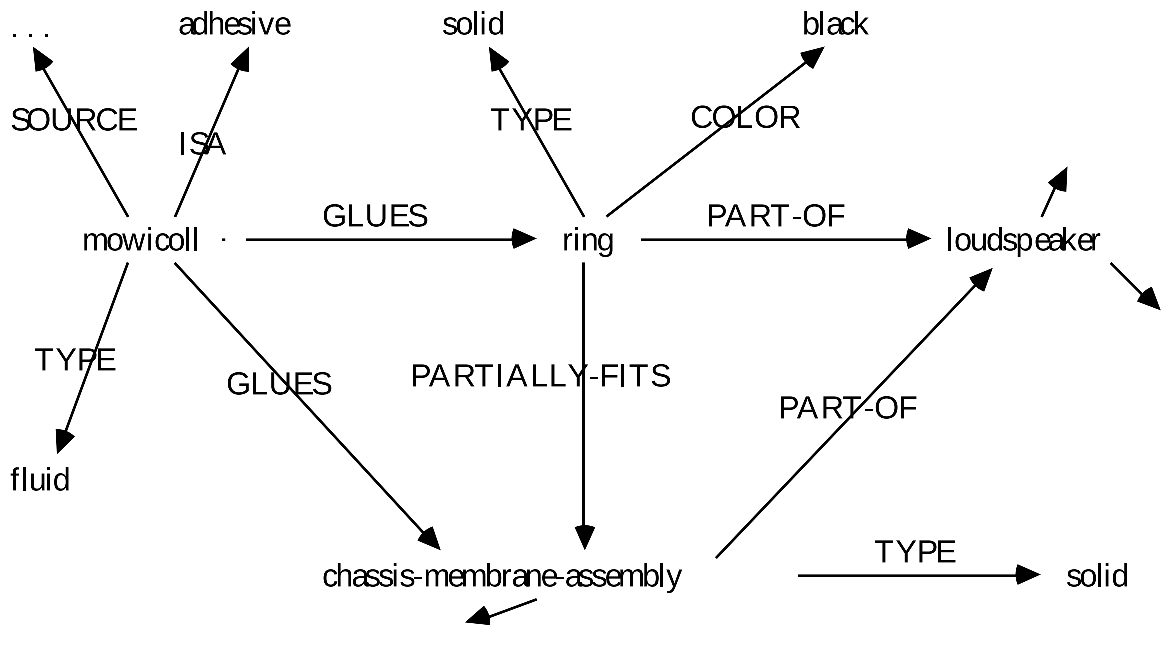


Figure 7.1. A hierarchical semantic network containing explicit representations of object properties and relations.

Other properties and relations may be implicitly specified by using inference rules for deducing them from other properties and relations, as indicated in figure 7.2.

∀x ∀y [(x GLUED-ON y) ∅ (x ATTACHED-ON y)]
∀x ∀y ∀z [(z ISA adhesive) & (z GLUES x) & (z GLUES y) &
(z BETWEEN x y) ∅ (x GLUED-ON y)]
∀x ∀y [(x GLUES y) ∅ (x ADHERENT-ON y)]

Figure 7.2. Inference rules for deducing new properties and relations of objects.

The action models describe the actions that may be performed in the domain.

Each 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 7.3 presents the models of the actions from the problem solving episode in figure 6.2.

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 7.3 Action models.

7.2 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 [Fikes & al. 1972], [DeJong & Mooney, 1986], [Mitchell & al. 1986] and others.

First, one proves 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 which will be present in the condition of General Rule 1.

Secondly, one generalizes the proof tree as much as possible so that the proof still holds.

In this way one generalizes the problem, its solution, and the relevant features.

Thirdly, one formulates 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.

7.3 Proving the example

Let us consider again **Example 1** (see figure 6.2).

To prove this example means to show that the sequence of the actions

APPLY OBJECT mowicoll ON ring

PRESS OBJECT ring ON chassis-membrane-assembly

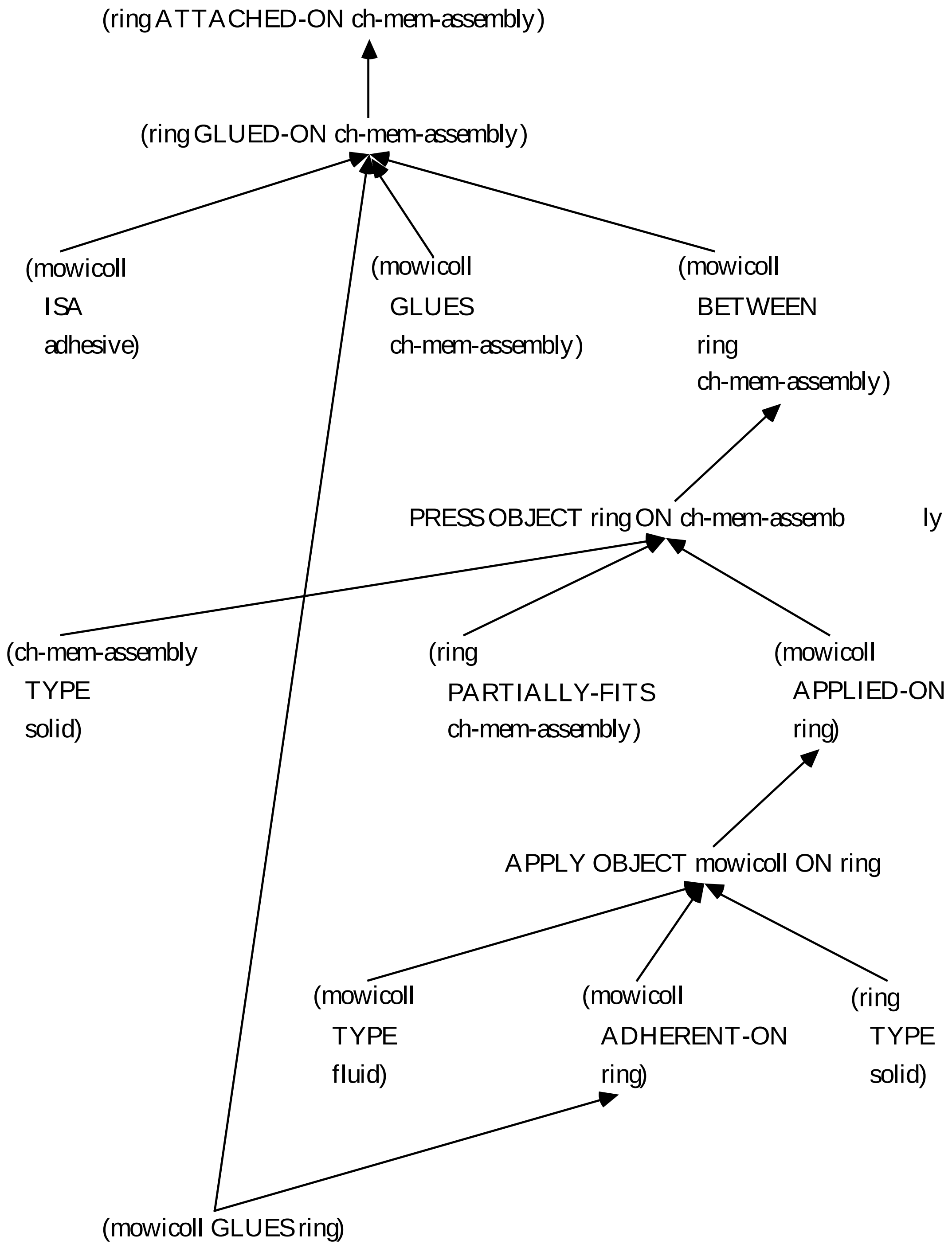
achieves the goal of the action

ATTACH OBJECT ring ON chassis-membrane-assembly

This goal is:

(ring ATTACHED-ON chassis-membrane-assembly)

The proof is indicated in figure 7.4. It was obtained by using the object descriptions in figure 7.1, the inference rules in figure 7.2, and the action models in figure 7.3.



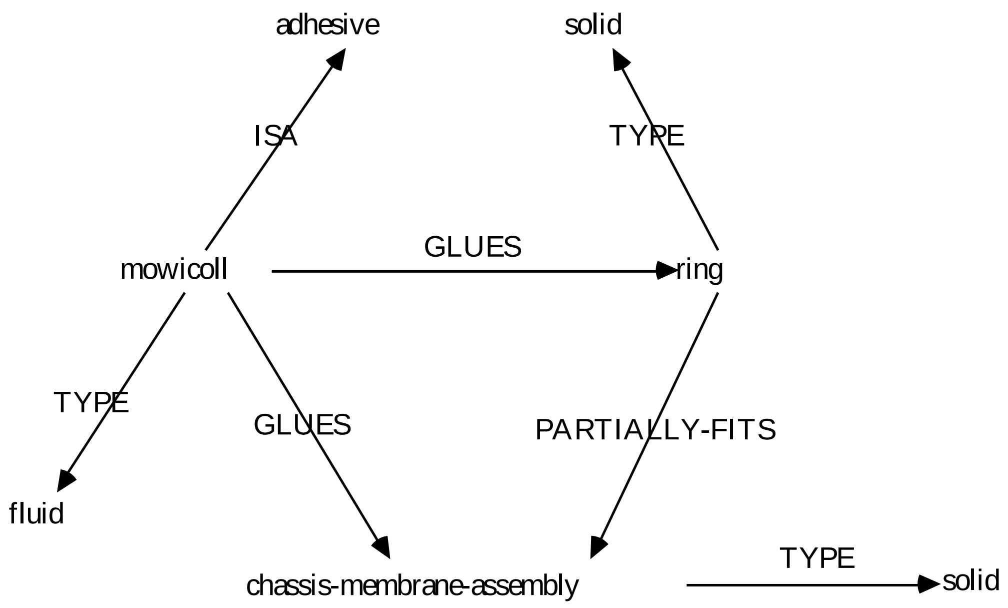
(ring ATTACHED-ON ch-mem-assembly)
(ring GLUED-ON ch-mem-assembly)
(mowicoll ISA adhesive)
(mowicoll GLUES ch-mem-assembly)
(mowicoll BETWEEN ring ch-mem-assembly)
PRESS OBJECT ring ON ch-mem-assembly
(ch-mem-assembly TYPE solid)
(ring PARTIALLY-FITS ch-mem-assembly)
(mowicoll APPLIED-ON ring)
APPLY OBJECT mowicoll ON ring
(mowicoll TYPE fluid)
(mowicoll ADHERENT-ON ring)
(ring TYPE solid)
(mowicoll GLUES ring)

Figure 7.4. A complete proof of Example 1.

In the above proof, 'ch-mem-assembly' stands for 'chassis-membrane-assembly'.

The leaves of this tree are those features of 'ring', 'chassis-membrane-assembly', and 'mowicoll' which 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:



(ring TYPE solid) & (chassis-membrane-assembly TYPE solid) &
(ring PARTIALLY-FITS chassis-membrane-assembly) &
(mowicoll ISA adhesive) & (mowicoll TYPE fluid) &
(mowicoll GLUES chassis-membrane-assembly) &
(mowicoll GLUES ring)

Figure 7.5. The relevant features of Example 1.

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.

7.4 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 7.4 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 & Bennet, 1986].

First, one replaces the instances of the action models and inference rules in figure 7.4 with their general patterns (taken from the figures 7.2 and 7.3), thus obtaining the proof structure in figure 7.6.

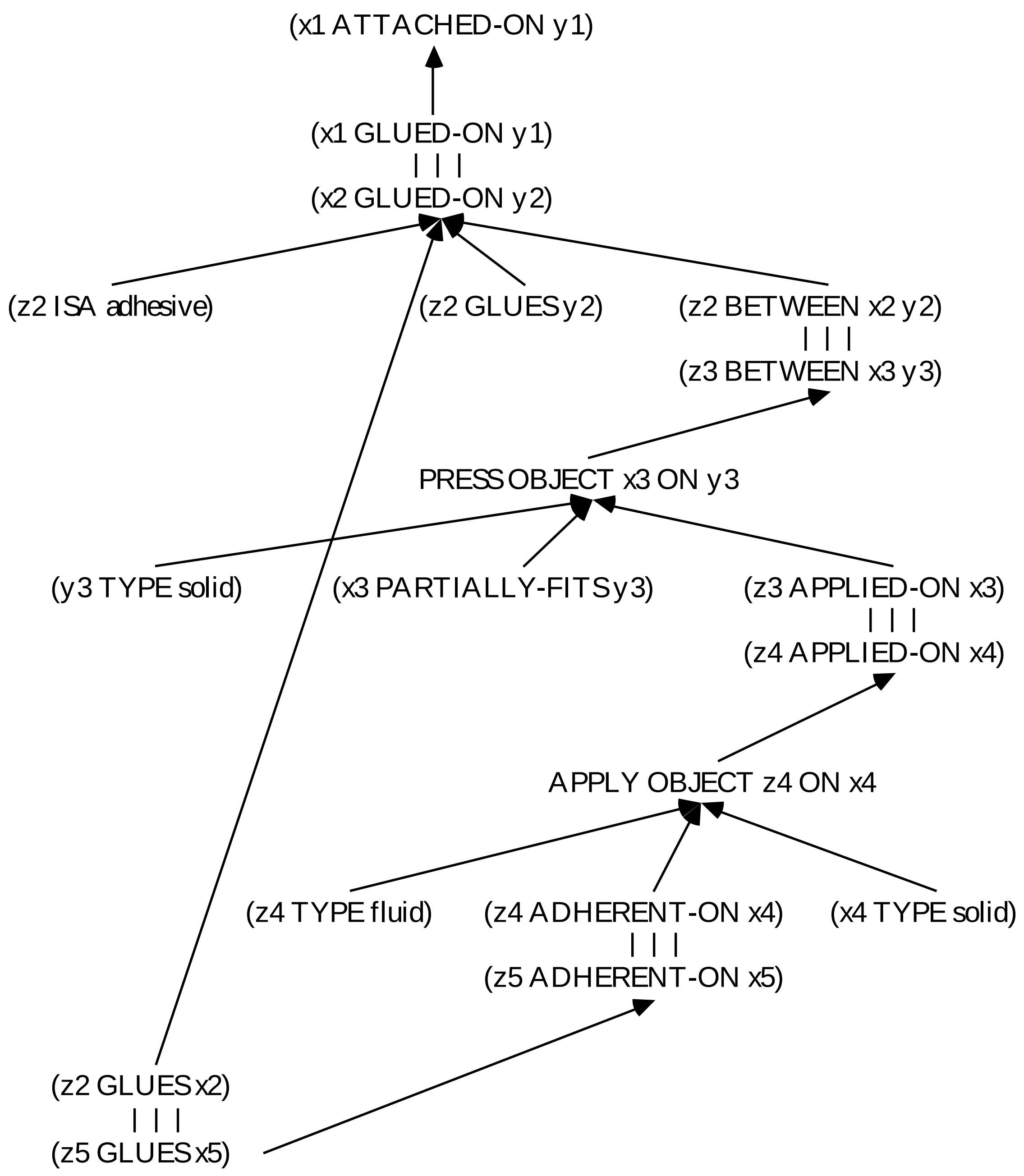


Figure 7.6. The general structure of the proof.

Secondly, one determines the most general unifying substitution for each connection pattern (the patterns connected by |||) and composes these substitutions, as indicated in figure 7.7.

```
(x1 GLUED-ON y1)      (x2 <– x1, y2 <– y1)
     |||
(x2 GLUED-ON y2)

(z2 GLUED-ON x2)      (x2 <– x1, y2 <– y1,
     |||               z5 <– z2, x5 <– x2)
(z5 GLUED-ON x5)

(z2 BETWEEN x2 y2)    (x2 <– x1, y2 <– y1,
     |||                z5 <– z2, x5 <– x2,
(z3 BETWEEN x3 y3)     z3 <– z2, x3 <– x2, y3 <– y2)


(z3 APPLIED-ON x3)    (x2 <– x1, y2 <– y1,
     |||                z5 <– z2, x5 <– x2,
(z4 APPLIED-ON x4)     z3 <– z2, x3 <– x2, y3 <– y2,
                        z4 <– z3, x4 <– x3)

(z4 ADHERENT-ON x4)   (x2 <– x1, y2 <– y1,
     |||                z5 <– z2, x5 <– x2,
(z5 ADHERENT-ON x5)    z3 <– z2, x3 <– x2, y3 <– y2,
                        z4 <– z3, x4 <– x3,
                        z5 <– z4, x5 <– x4)
```

Figure 7.7. The computation of the unifying substitution.

Finally, one applies the composed substitution to the structure in figure 7.6, thus obtaining the generalized proof in figure 7.8.

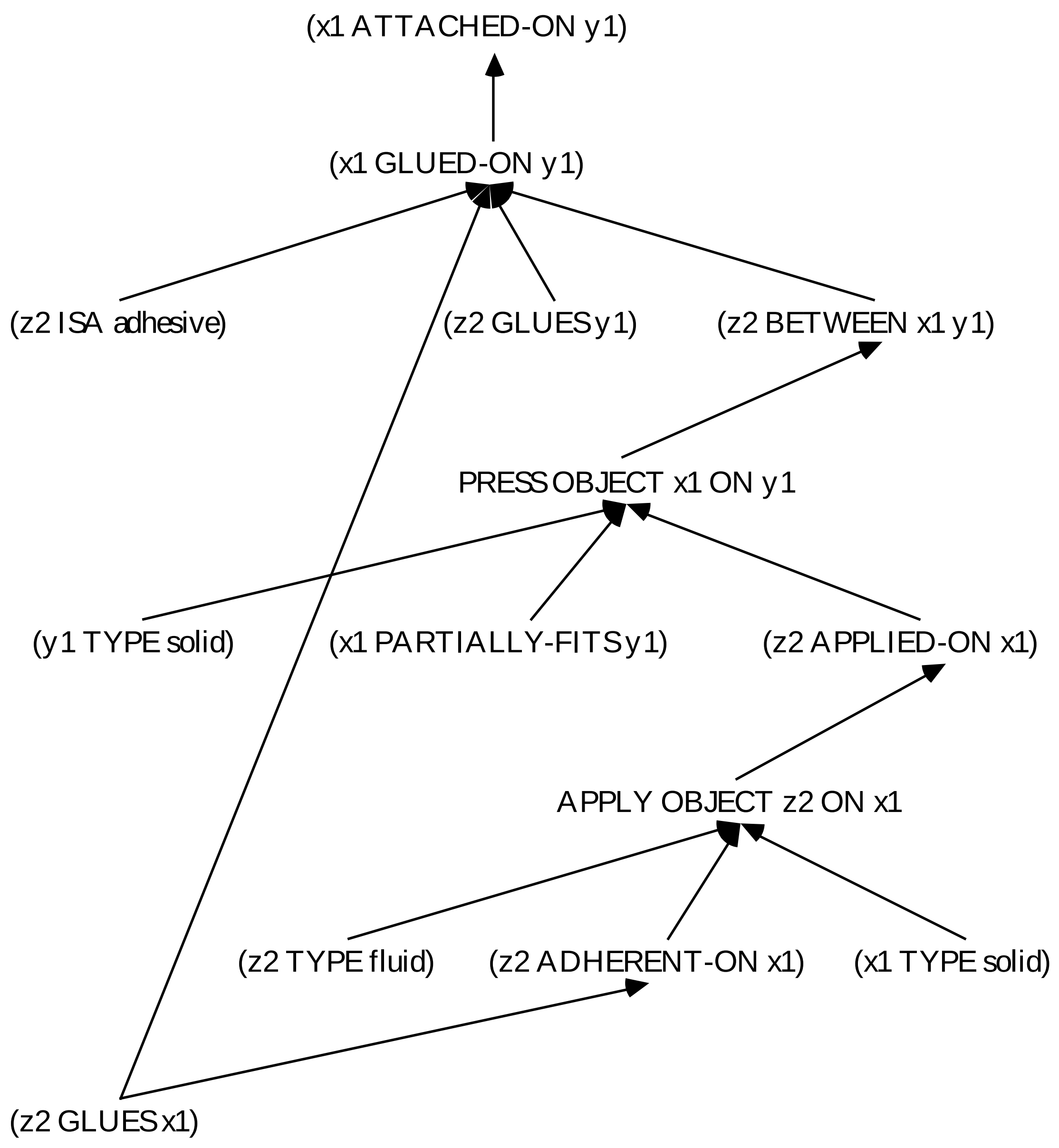


Figure 7.8. The generalization of the proof in figure 7.4.

7.5 The formulation of the general rule

Just for the purpose of facilitating the comparison with the other learning methods of DISCIPLE, let us apply to the generalized proof in figure 7.8 the substitution

(x1<–x, y1<–y, z2<–z)

which does not change its generality:

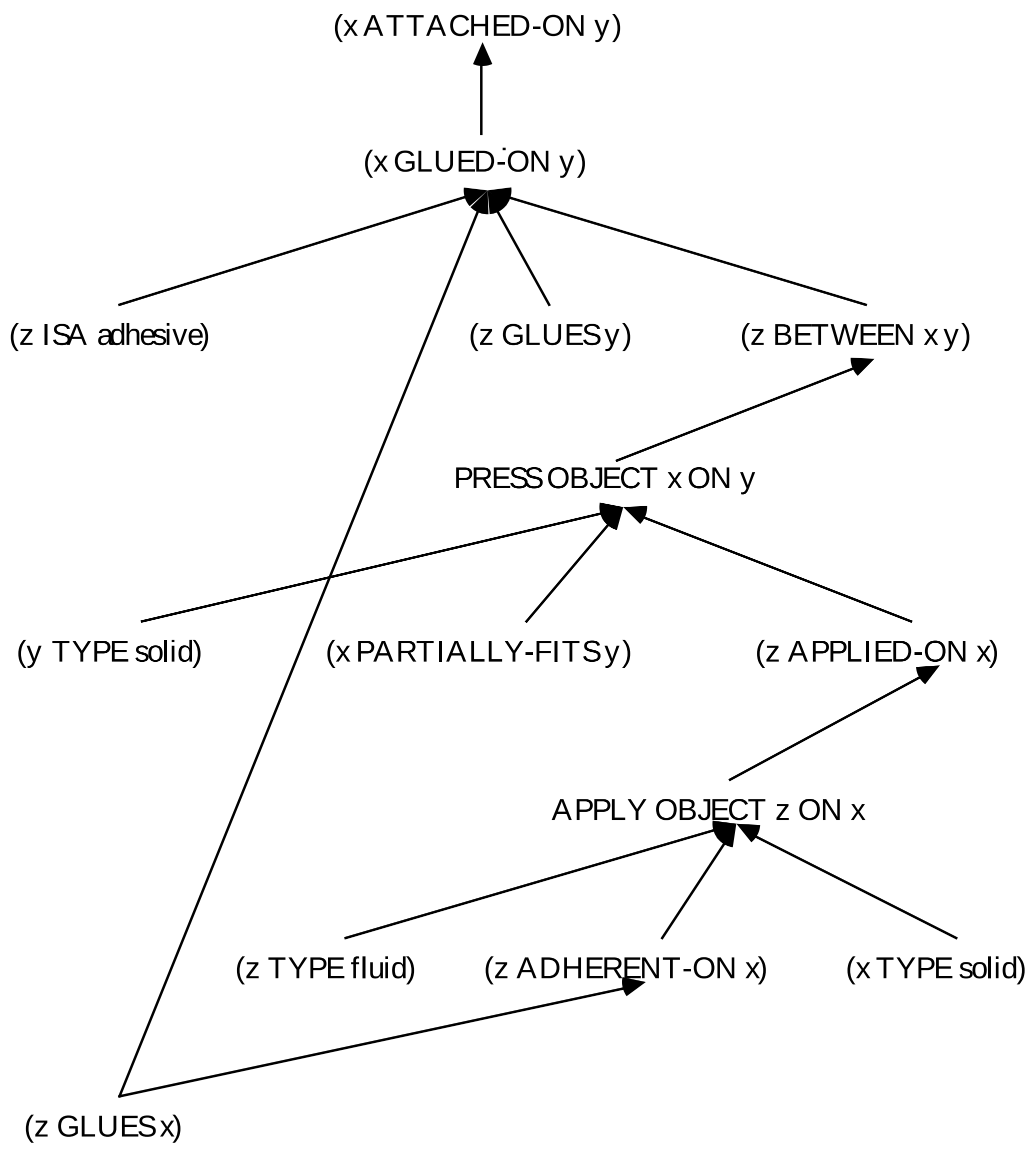


Figure 7.9. Equivalent form of the generalized proof in figure 7.8.

The leaves of this generalized tree represent a justified generalization of the relevant features in figure 7.5.

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

Figure 7.10. Justified generalization of the relevant features of Example 1.

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

This may be expressed in the form of the following general decomposition rule, which is precisely the decomposition rule learned from **Example 1**:

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 7.11. The decomposition rule learned from Example 1.

8. LEARNING IN A WEAK THEORY DOMAIN

8.1 A sample of a weak theory

A weak theory about the problem solving episode in figure 6.2 (Example 1) consists of the (possible incomplete) descriptions of the objects from this episode. It does not contain the models of the actions from this episode.

A sample of such a theory is represented in figure 8.1.

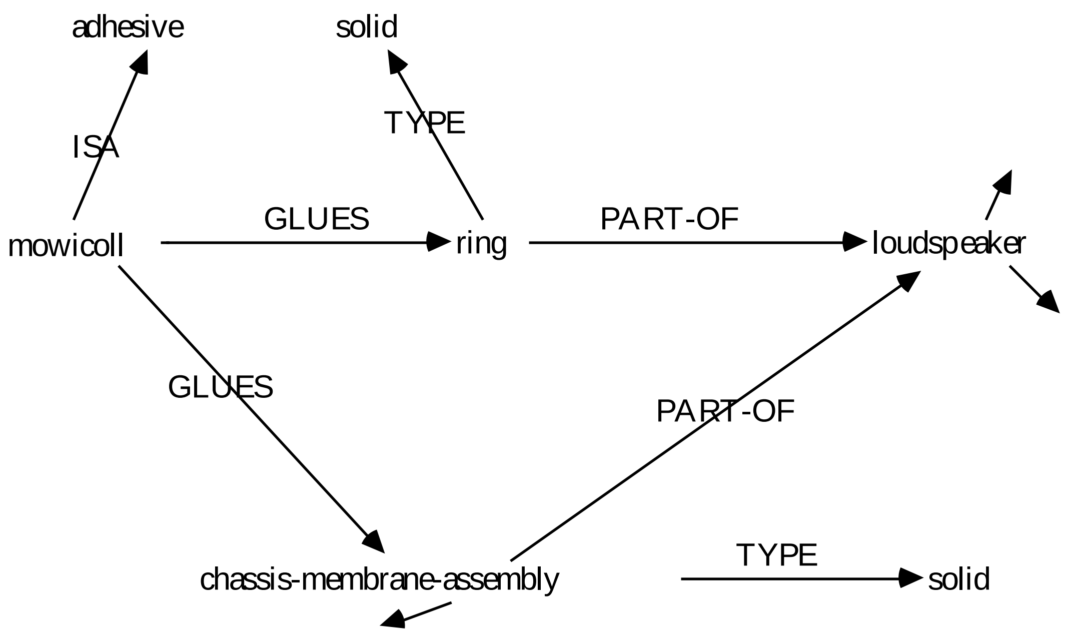


Figure 8.1. Fragment of a weak theory.

The idea of considering such a theory came from our experience with building a knowledge base for loudspeaker manufacturing.

We realized that it was very difficult for our expert (Zani Bodnaru from the Industrial Electronic Enterprise in Bucharest) to describe the actions in terms of their preconditions and effects.

On the contrary, it was much easier for him to describe the objects and to give us examples of decompositions and specializations.

This should not surprise anyone working in action planning who knows the difficulty of defining action models for complex operations.

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

8.2 General presentation of the learning method

In the context of a weak theory, the system will try to compensate its 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 8.2.

First DISCIPLE looks for a shallow explanation of 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 which are analogous with the user's example. DISCIPLE carefully 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.

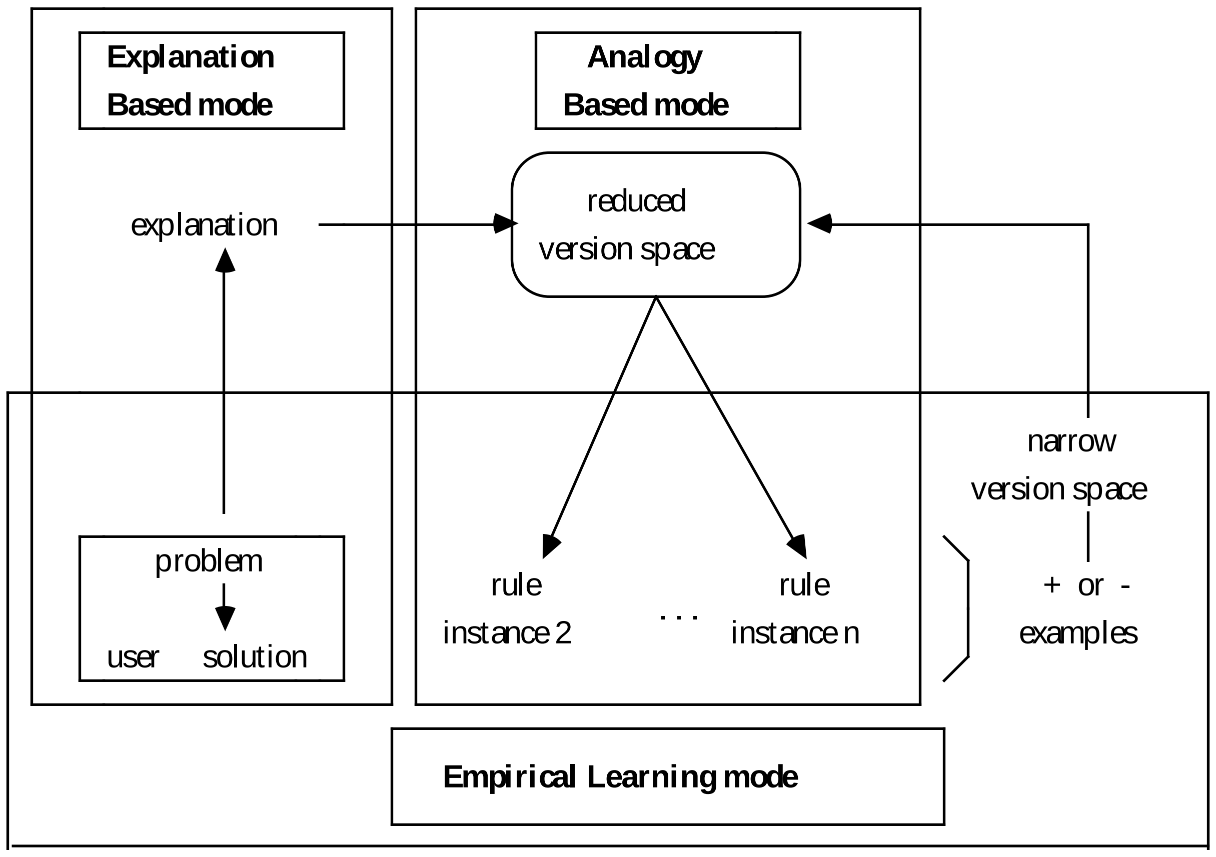


Figure 8.2 The learning method in the context of a weak theory.

In the following sections we shall present in detail this learning method by using again **Example 1** from figure 6.2.

8.3 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 6.2.

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.

8.3.1 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 7, 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

Let us consider the problem solving episode in figure 6.2 (Example 1). In the context of a weak theory, a complete explanation of this problem solving episode is the following one:

(ring TYPE solid) & (chassis-membrane-assembly TYPE solid) &
(ring PARTIALLY-FITS chassis-membrane-assembly) &
(mowicoll ISA adhesive) & (mowicoll TYPE fluid) &
(mowicoll GLUES chassis-membrane-assembly) &
(mowicoll GLUES ring)

Figure 8.3. A complete explanation of Example 1.

The fact that the 'ring', the 'chassis-membrane-assembly', and the 'mowicoll' have the features in figure 8.3 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.

As can be seen, this explanation consists of the leaves of the tree in figure 7.4. Since such a proof tree cannot be built in a weak theory, one has to use heuristics, as well as the user's help, in order to find the explanation.

Moreover, it is very likely that one will not find the complete explanation in figure 8.3. 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).

8.3.2 A heuristic to find explanations

In a weak theory domain, the explanation of Example 1 is the result of an interactive process in which the system uses heuristics to propose plausible partial explanations to be validated by the user who may himself indicate other pieces of explanations.

In DISCIPLE, this process of finding the explanation is based on the following heuristic: *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 ? <u>Yes</u>
mowicoll GLUES chassis-membrane-assembly ? <u>Yes</u>
ring PART-OF loudspeaker &
chassis-membrane-assembly PART-OF loudspeaker ? <u>No</u>

All the pieces of explanations marked by a user's yes form the explanation of the example rule:

Explanation 1:

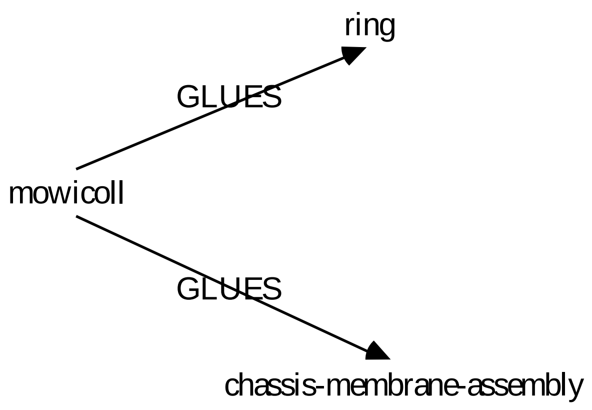


Figure 8.4. The explanation of Example 1.

Notice that this explanation is incomplete. As already stated, this is due partly to the incompleteness of the domain theory and partly to the heuristic used to find explanations (DISCIPLE looks only for the relations between objects, ignoring their properties). However, the found explanation 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 above presented heuristic.

8.4 Analogy-Based Mode

8.4.1 Learning by Analogy

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.

A general scenario for learning by analogy is expressed by the following statement:

'a T is like a B'

The purpose of this statement is to convey knowledge from 'B' to 'T'. 'B' is called the base since it is the entity that serves as a source of knowledge, and 'T' is called the target since it is the entity that receives the knowledge.

A classical example of analogy is Rutherford's analogy:

"The hydrogen atom is like our solar system"

By analogy with the solar system, one is able to get new knowledge about the hydrogen atom.

Let us notice that the base and the target are similar but not identical. Therefore, nothing guarantees that the features imported from the base are indeed features of the target. Otherwise stated, analogy is a weak inference method and the inferences drown by analogy have to be, somehow, validated.

According to the **structure-mapping theory** of Gentner [Gentner, 1983], which will be briefly presented in the following, the relations between objects, rather than attributes of objects, are mapped from the base to the target. Moreover, a relation that belongs to a mappable system of mutually interconnecting relationships is more likely to be imported into the target than is an isolated relation *(the systematicity principle).*

The analogy maps the objects of the base onto the objects of the target: b1–> t1, b2 –> t2, ... , bn –> tn

These object correspondences are used to generate the candidate set of inferences in the target domain.

Predicates from the base are carried across to the target, using the node substitutions dictated by the object correspondences, according to the following rules:

1. Discard attributes of objects

A(bi) -/-> A(ti)

For instance, the yellow color of the sun is not transmitted to the hydrogen nucleus.

2. Try to preserve relations between objects

R(bi, bj) -?-> R(ti, tj)

That is, some relations are transmitted to the target, while others are not.

3. The systematicity principle: the relations that are most probably to be transmitted are those belonging to systems of interconnected relations

R'(R1(bi,bj), R2(bk,bl)) –> R'(R1(ti,tj), R2(tk,tl))

An important result of the learning by analogy research ([Bareiss & Porter 1987], [Burstein 1986], [Carbonell, 1983, 86], [Chouraqui, 1982], [Forbus & Gentner, 1986], [Kedar-Cabelli, 1985], [Russel, 1987], [Winston, 1980, 86]), confirming the structure-mapping theory, is that the analogy involves mapping some underlying causal network of relations between analogous situations.

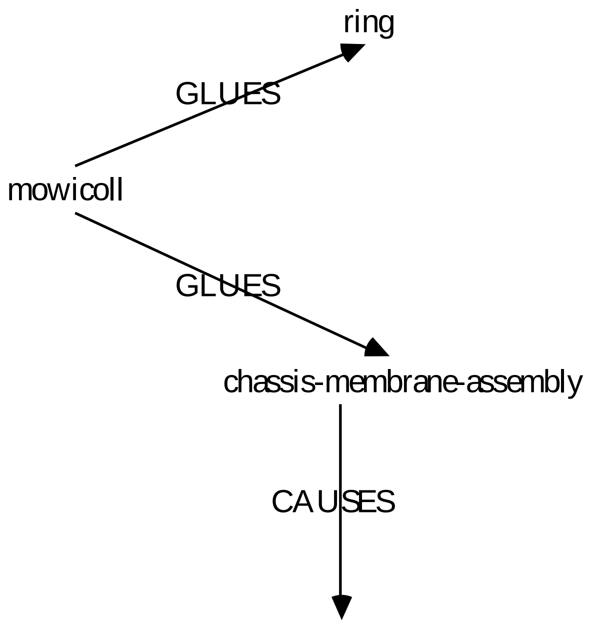
The idea is that similar causes are expected to have similar effects:

$$\begin{tikzcd} A \arrow[r, leftrightarrow, "\text{similar}"] \arrow[d, "\text{cause}"] & A' \arrow[d, "\text{cause}"] \\ B \arrow[r, leftrightarrow, "\text{similar}"] & B' \end{tikzcd}$$

Since A, B, A', B' are usually networks of relations, the 'cause' relation is a higher order relation. Therefore, this is in accordance with the systematicity principle.

8.4.2 The paradigm of analogy in DISCIPLE

In DISCIPLE, the explanation of a problem solving operation may be regarded as a cause for performing the operation:



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-assemby

Let us suppose that a new situation is characterized by the following network:

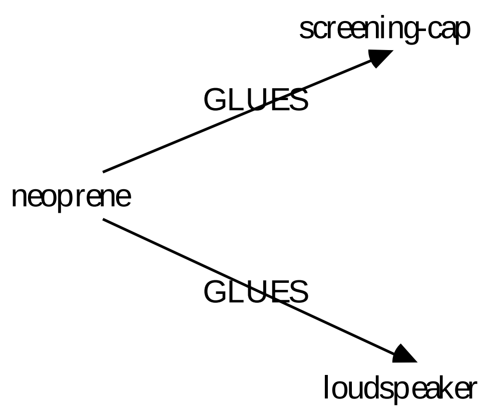


Figure 8.5. Network similar with **Explanation 1** (figure 8.4).

Since this network is similar with Explanation 1, we may expect that it will cause a decomposition of the problem

ATTACH OBJECT screening-cap ON loudspeaker

similar with the one from **Example 1**:

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

Now, let us point out that the analogical decomposition is not derived from any particular properties of 'screening-cap', loudspeaker', and 'neoprene' other than those that 'neoprene' glues both the 'screening-cap' and the 'loudspeaker'. Therefore, the system making this inference must be equally willing to infer the decomposition

Solve the problem
ATTACH OBJECT x ON y
by solving the subproblems
APPLY OBJECT z ON x
PRESS OBJECT x ON y

for any other objects 'x', 'y', and 'z', such that the following network holds:

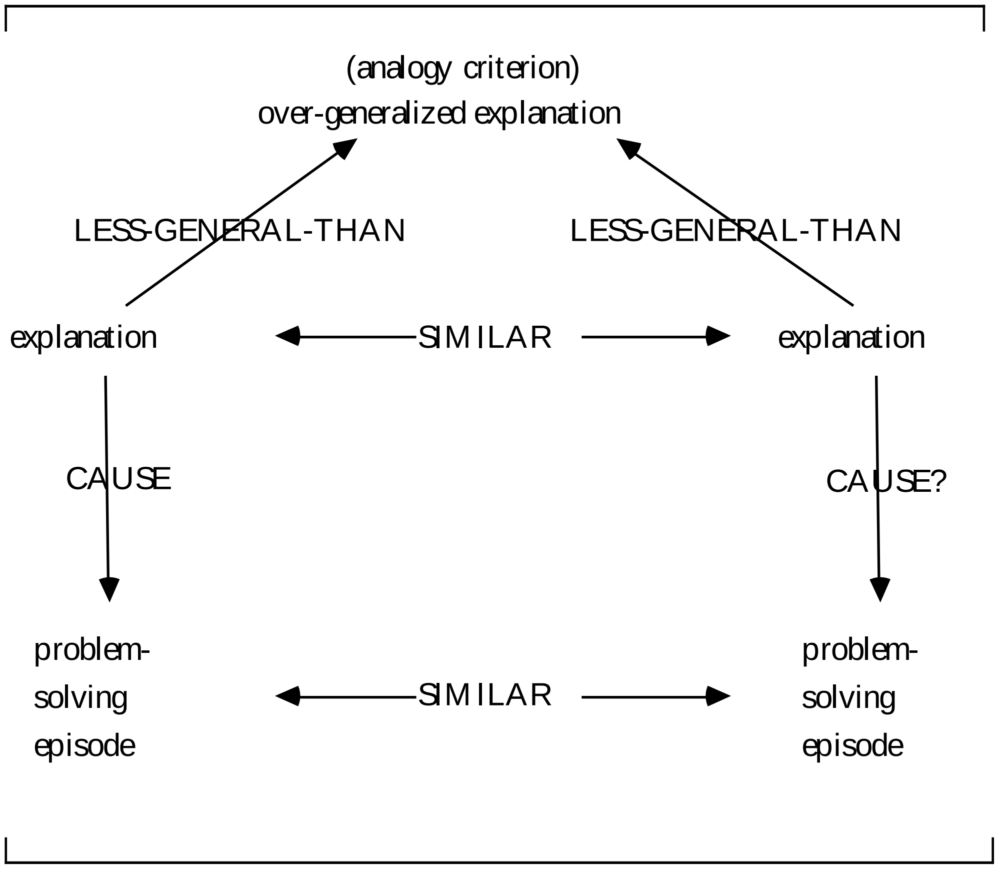
$$\begin{tikzcd}
 & & x \\
z \arrow[urr, "GLUES"] \arrow[drr, "GLUES"'] & & \\
 & & y
\end{tikzcd}$$

Therefore, this network may be regarded as an **analogy threshold,** and the above inferences may be rewritten in the following equivalent form:

```
IF
        analogy threshold
        (z GLUES x) & (z GLUES y)
THEN
solve the problem
        ATTACH OBJECT x ON y
by solving the subproblems
        APPLY OBJECT z ON x
        PRESS OBJECT x ON y
```

The instances of this rule are decompositions of the ATTACH operation that are analogous with Example 1. Since analogy is a weak inference method, these decompositions may be acceptable (i.e. valid decompositions) or not.

Therefore, the analogy paradigm in DISCIPLE is the following one:



The following figure contains an example of analogy.

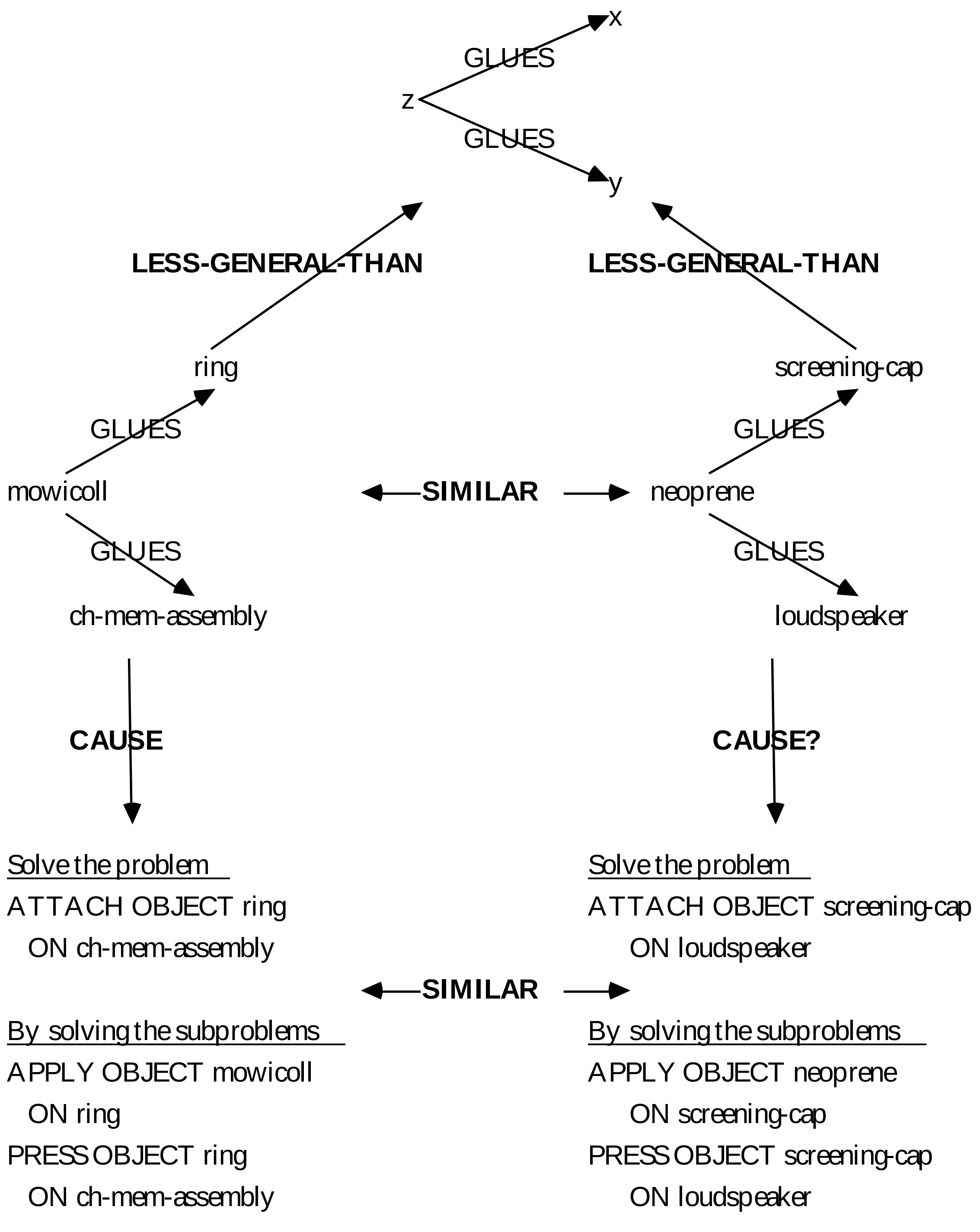


Figure 8.6. An example of analogy.

While, usually, two situations are considered to be analogous if they match within a certain pre-defined threshold, in DISCIPLE, two situations are considered to be analogous if they generalize within a pre-defined threshold (the analogy criterion). This is not at all surprising since generalization may be reduced to structural matching [Kodratoff & Ganascia, 1986].

Now we can also justify the heuristic used by DISCIPLE to find explanations of examples. Because these explanations are used in analogical reasoning, the *systematicity principle* requires them to be networks of relations. Indeed, in this case, the 'CAUSE' relation, which is imported by analogy, is a higher order relation.

8.4.3 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. This space contains rules covering only instances analogous with Example 1.

First of all DISCIPLE over-generalizes 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:

(x ISA ring) & (y ISA chassis-membrane-assembly) &
(z ISA mowicoll) & (z GLUES x) & (z GLUES y)

Figure 8.7 Explanation 1 written as a lower bound of the applicability condition of General Rule 1.

Notice that the above expression 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 which 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 8.7, by dropping the 'ISA' predicates.

In general, *the analogy criterion is defined as the most general generalization of Explanation 1 that may still be accepted by the user as an explanation of General Rule 1.*

The analogy criterion may be taken as an upper bound for the applicability condition of General Rule 1:

analogy criterion:
(z GLUES x) & (z GLUES y)

Figure 8.8. An over-generalization of Explanation 1.

The analogy criterion, Explanation 1, and the General Rule 1 define a reduced version space [Mitchell, 1978] for the rule to be learned:

IF

G:analogy criterion

(z GLUES x) & (z GLUES y)

S: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 8.9. A reduced version space for the rule to be learned.

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

8.4.4 Generation of instances

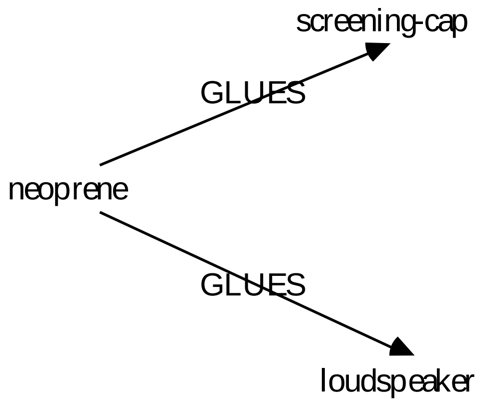
To search the rule in the space from figure 8.9, DISCIPLE needs positive and negative instances of it. *These instances may be provided by future problem solving episodes or may be generated by the system itself.*

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:

Explanation-i:



It uses the found objects to generate an instance of General Rule 1 (see figure 8.9) and asks the user to validate it:

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 ?

Figure 8.10. An instance generated by analogy with Example 1.

8.5 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 8.9.

8.5.1 The use of the positive examples

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

Let us suppose that the user accepts the decomposition in figure 8.10. Then, Explanation-i, computed in the previous section, is a true explanation which 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 8.9 and Explanation-i:

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

This generalization is taken 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
```

Figure 8.11. The version space after the use of a new positive example.

Notice 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.

8.5.2 The use of the negative examples

Each negative example shows the incompleteness of Explanation 1 and of its over-generalization (the analogy criterion). The explanation of why the instance is a negative example points to the features which 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'. They also satisfy the analogy criterion (the upper bound of the condition version space) but the corresponding instance is rejected by the user:

May I solve the problem
    ATTACH OBJECT screening-cap ON loudspeaker
by solving the subproblems
    APPLY OBJECT scotch ON screening-cap
    PRESS OBJECT screening-cap ON loudspeaker ? <u>No</u>

Figure 8.12. A negative example of the rule to be learned.

In this case, DISCIPLE looks for an explanation of the failure because this explanation points to the important object features which were not contained in Explanation 1.

The explanation is that 'scotch' (an adhesive tape) is not fluid (therefore, it might not be applied on a curved surface):

Failure Explanation:
NOT (scotch TYPE fluid)

Figure 8.13. The explanation of the negative example in figure 8.12.

That is, the concept represented by 'z' must not have the following property:

'NOT (z TYPE fluid)'

Therefore, DISCIPLE will specialize both bounds of the version space by adding the negation of this explanation:

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

Figure 8.14. The version space after the use of a negative example.

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 7.11.

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).

It should be noticed that the degree of generalization of the learned rule is determined by the degree of generalization of the over-generalized explanation in figure 8.8. However, this rule may still be generalized in the future. This opportunity arises when the user indicates that two objects may be attached by a gluing process even if the rule does not apply. In such a case, the conditions of the rule may be generalized, so that to cover the explanation of the new example.

8.5.3 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.

DISCIPLE's 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 whose answers allow it to take the right decision.

This process is illustrated in section 8.8.

8.6 Developing the domain theory

As has been shown in section 8.3.2, 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 failure explanation in figure 8.13 was provided by the user. In this case the domain theory will be enriched by storing this explanation:

NOT (scotch 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 are added to the domain theory:

(mowicoll TYPE fluid)

(neoprene TYPE fluid)

8.7 The learning algorithm

The purpose of the previous sections was to justify the following learning algorithm.

Explanation-Based Mode:

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

Analogy-Based Mode:

2. Over-generalize 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. Over-generalize 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 over-generalized 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 "interesting" objects satisfying the Upper Bound.

If there are such objects then Call Explanation-i the properties of these objects which 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 with 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, which is less general than the Upper Bound. Several cases may occur:

- if such a generalization exists and is not identical with the Upper Bound, then take it as the new Lower Bound and go to step 6;
- if such a generalization exists and 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.

8.8 A sample trace of the learning algorithm

In this section we shall apply the learning algorithm in order to learn a specialization rule.

Let us consider the following example of specialization:

Example 1:

Solve the problem

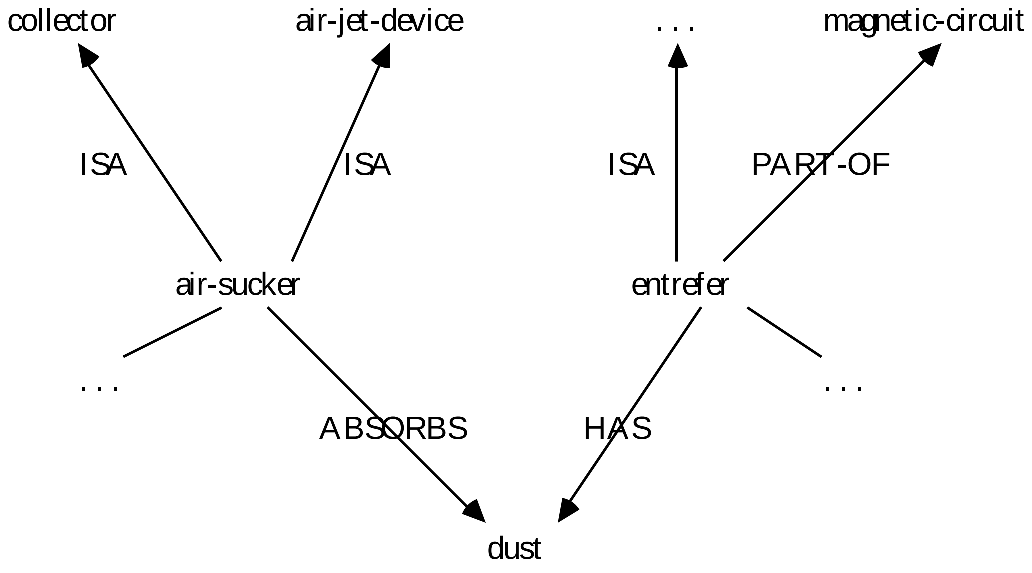
CLEAN OBJECT entrefer

by solving the specialization

CLEAN OBJECT entrefer WITH air-sucker

1. Find an explanation of Example 1

Look, in the knowledge base, for the links (or paths) connecting 'entrefer' and 'air-sucker'. They are illustrated in the following network:

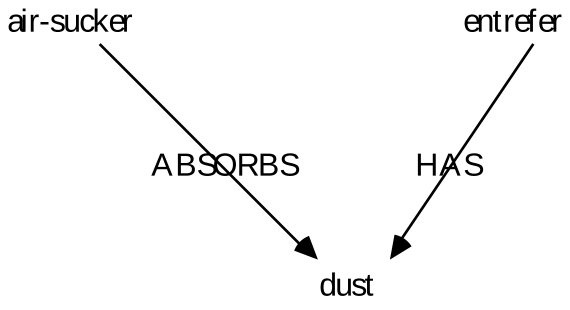


This network contains a single path between 'entrefer' and 'air-sucker'. This is proposed to the user as a plausible explanation of Example 1:

Do the following justify your solution:
entrefer HAS dust & air-sucker ABSORBS dust ? *Yes*

Therefore, the explanation of Example 1 is:

Explanation 1:



2. Build General Rule 1

General rule 1:
solve the problem
CLEAN OBJECT x
by solving the specialization
CLEAN OBJECT x WITH y

3. Rewrite Explanation 1 as a Lower Bound for the applicability condition of General Rule 1

Lower Bound:
(x ISA entrefer) & (y ISA air-sucker) & (z ISA dust) &
(x HAS z) & (y ABSORBS z)

4. Over-generalize Explanation 1

DISCIPLE is first trying to generalize Explanation 1 by generalizing the contained relations only.

It will show the user the competing generalizations, asking him to choose the right one:

The explanation
entrefer HAS dust & air-sucker ABSORBS dust
may be expressed as:
1.entrefer HAS dust & air-sucker REMOVES dust
2.entrefer HAS dust & air-sucker GETS dust
Choose the solution [number, No, Modify]: 1

Therefore, a first generalization of Explanation 1 is:

entrefer HAS dust & air-sucker REMOVES dust

Next, the above explanation is over-generalized by turning all the objects into variables, thus obtaining:

Over-generalized explanation:
(x HAS z) & (y REMOVES z)

5. Build the reduced version space for the rule to be learned

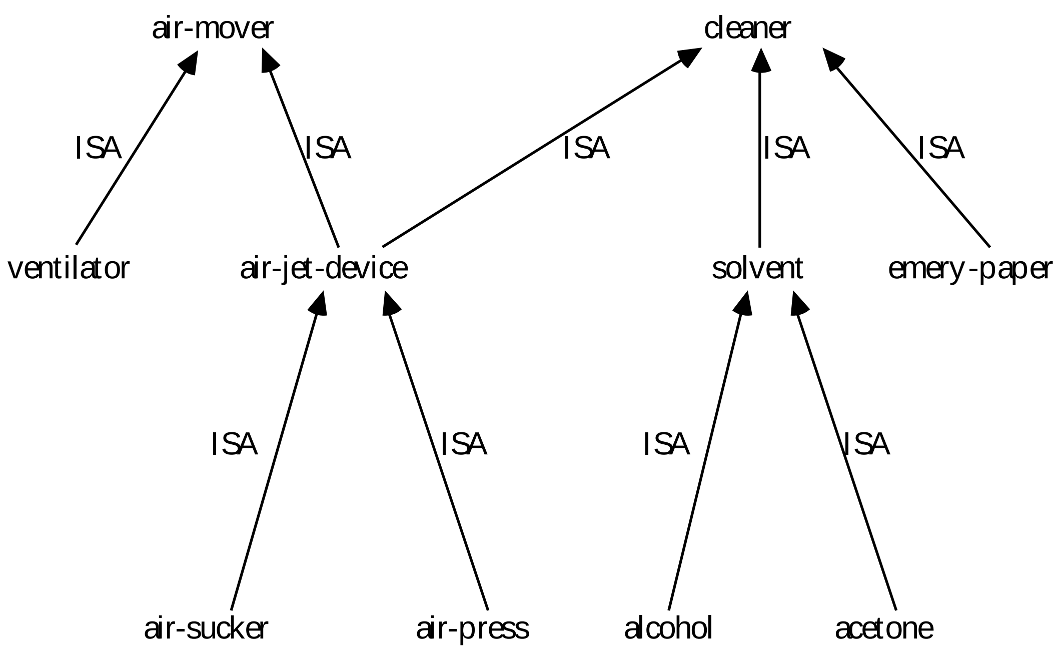
IF
Upper Bound:
(x HAS z) & (y REMOVES z)

Lower Bound:
(x ISA entrefer) & (y ISA air-sucker) &
(z ISA dust) & (x HAS z) & (y ABSORBS z)
THEN
General rule 1:
solve the problem
CLEAN OBJECT x
by solving the specialization
CLEAN OBJECT x WITH y

6. Look for objects satisfying the Upper Bound

As has been written in section 8.5.3, DISCIPLE is trying to generalize the Lower Bound by climbing the generalization hierarchies of the contained objects.

In our case, 'air-sucker' belongs to quite a rich generalization hierarchy:



Therefore, DISCIPLE is trying to generalize 'air-sucker' (in the Lower Bound) to its ancestors from the above hierarchy.

As may be seen, a first generalization consists in generalizing 'air-sucker' to 'air-jet-device'. This generalization is plausible since 'air-press', 'entrefer', and 'dust' satisfy the Upper Bound:

Explanation i:
(x ISA entrefer) & (y ISA air-press) & (z ISA dust) &
(x HAS z) & (y REMOVES z)

<u>7 & 8. Generate an instance analogous with Example 1 and propose it to the user</u>

May I solve the problem
CLEAN OBJECT entrefer
by solving the specialization
CLEAN OBJECT entrefer WITH air-press ? <u>*Yes*</u>

14 & 15. Generalize the Lower Bound in order to cover Explanation-i

Lower Bound:
(x ISA entrefer) & (y ISA air-jet-device) &
(z ISA dust) & (x HAS z) & (y REMOVES z)

6. Look for objects satisfying the Upper Bound

Now there are two possible generalizations of the 'air-jet-device': one to 'air-mover' and the other to 'cleaner'. To choose between them, DISCIPLE will build an instance based on an object belonging only to one of these competing generalizations. Such an object is, for instance, 'acetone', the corresponding Explanation-i being:

Explanation i:
(x ISA membrane-assembly) & (y ISA acetone) &
(z ISA surplus-glue) & (x HAS z) & (y DISSOLVES z)

7 & 8. Generate an instance analogous with Example 1 and
propose it to the user

May I solve the problem
CLEAN OBJECT membrane-assembly
by solving the specialization
CLEAN OBJECT membrane-assembly WITH acetone ? Yes

14 & 15. Generalize the Lower Bound in order to cover Explanation i

Lower Bound:
(x HAS z) & (y ISA cleaner) & (y REMOVES z) &
(z ISA waste-material)

6. Look for objects satisfying the Upper Bound

The 'air-jet-device' was generalized to 'cleaner'. However, this generalization is valid only if every son of 'cleaner' produces positive examples of General Rule 1. Such a son is 'emery-paper'. It satisfies the Upper Bound:

Explanation i:
(x ISA membrane-assembly) & (y ISA emery-paper) &
(z ISA surplus-glue) & (x HAS z) & (y REMOVES z)

Therefore, DISCIPLE may test the generalization of 'air-jet-device' to 'cleaner'.

<u>7 & 8. Generate an instance analogous with Example 1 and propose it to the user</u>

May I solve the problem
CLEAN OBJECT membrane-assembly
by solving the specialization
CLEAN OBJECT membrane-assembly WITH emery-paper ?
<u>No</u>

<u>9 & 10. Find an explanation of the negative example</u>

Since the user rejected the above specialization it follows that the concept represented by 'y' must not cover 'emery-paper', that is, it must be less general than 'cleaner'.

The fact that Explanation i is true is not enough to justify the above specialization.

In such a case, DISCIPLE re-enters the Explanation-Based Mode to find an explanation of why the above instance is a negative example. This explanation is the following one:

Failure Explanation i:
(emery-paper DESTROYS membrane-assembly)

<u>11 & 12. Specialize the Upper Bound and the Lower Bound so</u>

that not to cover Failure Explanation i

Failure-explanation-i shows that the concepts represented by 'x' and 'y' must not have the following property:

'(y DESTROYS x)'

Therefore, DISCIPLE will specialize both bounds of the version space by adding the predicate

'NOT(y DESTROYS x)'

Upper Bound:
(x HAS z) & (y REMOVES z) & NOT(y DESTROYS x)

Lower Bound:
(x HAS z) & (y ISA cleaner) & (y REMOVES z) &
(z ISA waste-material) & NOT(y DESTROYS x)

It may happen that the domain theory does not contain the explanation of the negative example. In such a case the user is asked to provide himself the explanation. As a side effect, the domain theory will be enriched by retaining the new link between 'emery-paper' and 'chassis-assembly'.

Moreover, as a consequence of updating the Lower Bound, the following relations between the objects that generated positive examples of the rule are added to the domain theory:

NOT(air-sucker DESTROYS entrefer)
NOT(air-press DESTROYS entrefer)
NOT(acetone DESTROYS membrane-assembly)

6. Look for objects satisfying the Upper Bound

The domain theory does not contain other "interesting" objects that would allow efficient improvement of the rule's conditions. Therefore, the learned rule is:

IF

  Upper Bound:

  (x HAS z) & (y REMOVES z) & NOT(y DESTROYS x)

  Lower Bound:

  (x HAS z) & (y ISA cleaner) & (y REMOVES z) &
  (z ISA waste-material) & NOT(y DESTROYS x)

THEN

General rule 1:

solve the problem

  CLEAN OBJECT x

by solving the specialization

  CLEAN OBJECT x WITH y

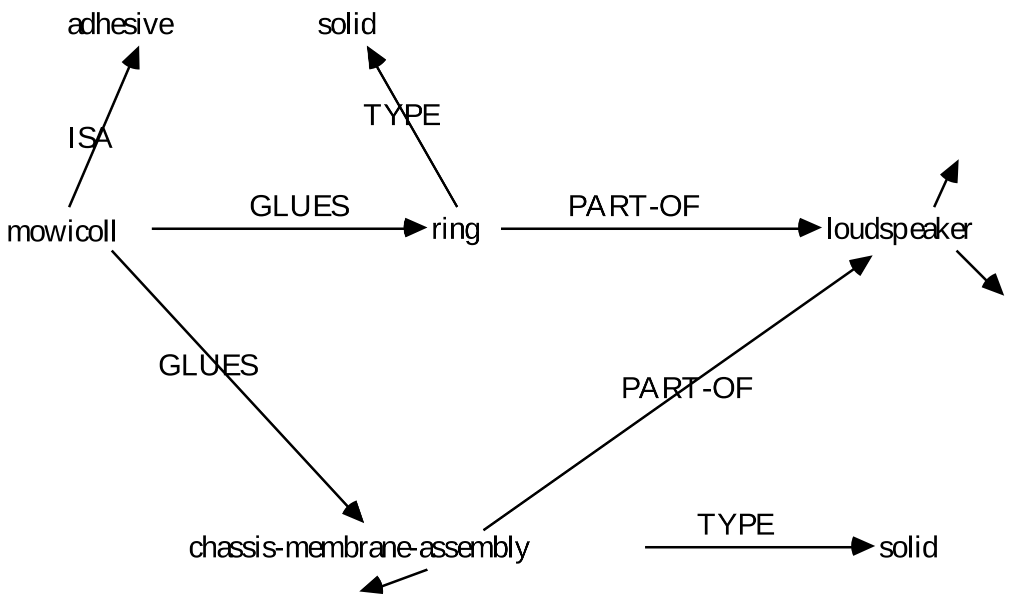
9. LEARNING IN AN INCOMPLETE THEORY DOMAIN

9.1 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.

A sample of an incomplete theory about Example 1 (figure 6.2) is given in the following figures:



∀x ∀y [(x GLUED-ON y) –> (x ATTACHED-ON y)]
∀x ∀y ∀z [(z ISA adhesive) & (z GLUES x) & (z GLUES y) &
(z BETWEEN x y) –> (x GLUED-ON y)]
∀x ∀y [(x GLUES y) –> (x ADHERENT-ON y)]

Figure 9.1. 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 9.2. Incomplete models of two actions from Example 1.

As one may notice, the explicit properties of the objects 'ring', 'chassis-membrane-assembly' and 'mowicoll' are the ones considered in the case of the weak theory (see figure 8.1).

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

9.2 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 the user's example and will generalize it, as in a complete theory.

In this way, the system will determine an **over-generalized explanation** of the example.

Then, the system will use the over-generalized explanation as an **analogy criterion** to perform experiments and to synthesize the general rule, as in a weak theory.

The first step may require asking focused questions to the user, in order to fill the possible gaps in the proof.

Also, the proof found in the first step will provide additional focus for the second step.

As a side effect of rule learning, one will develop the domain theory.

9.3. 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.

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:

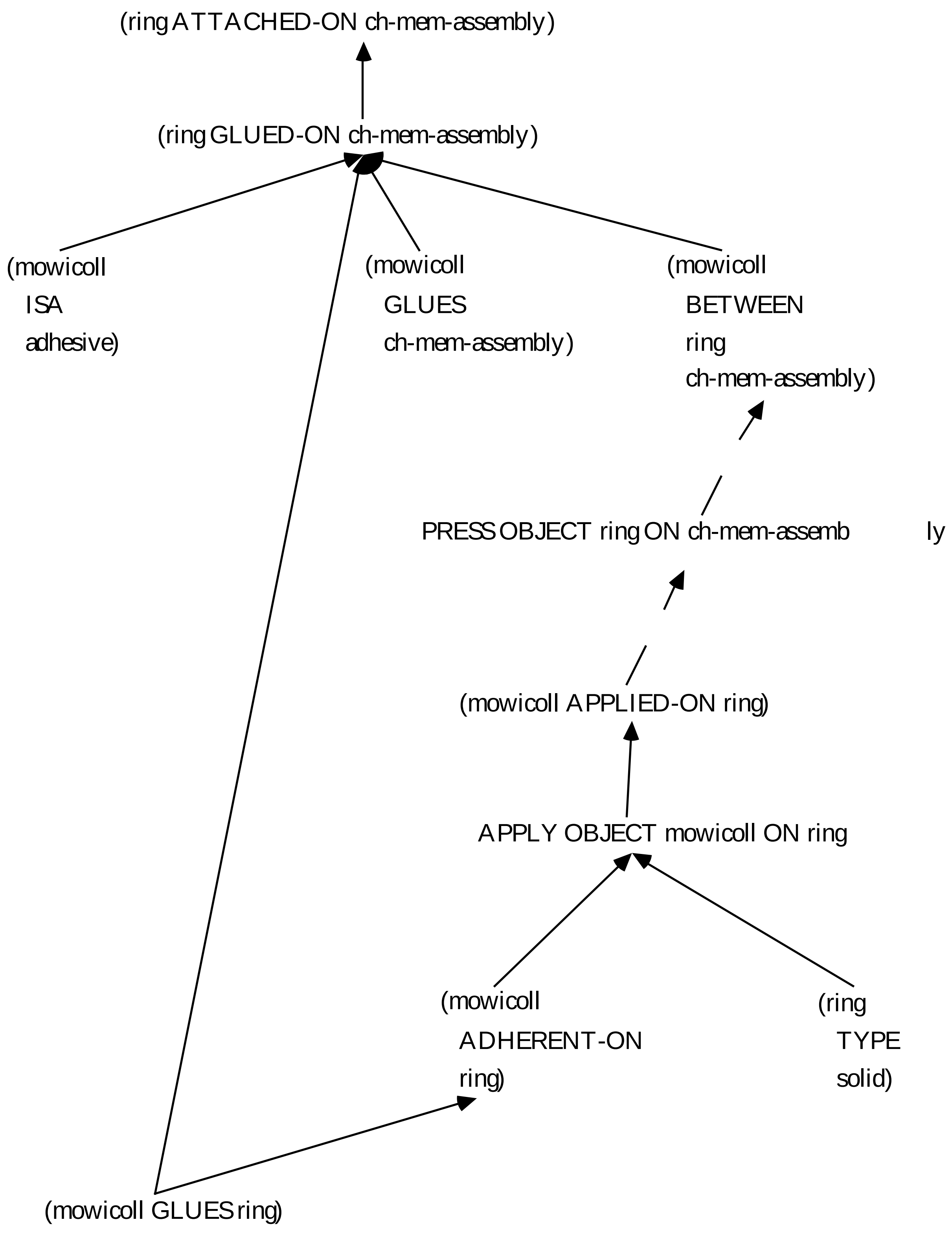

(ring ATTACHED-ON ch-mem-assembly)
(ring GLUED-ON ch-mem-assembly)
(mowicoll ISA adhesive)
(mowicoll GLUES ch-mem-assembly)
(mowicoll BETWEEN ring ch-mem-assembly)
PRESS OBJECT ring ON ch-mem-assemb ly
(mowicoll APPLIED-ON ring)
APPLY OBJECT mowicoll ON ring
(mowicoll ADHERENT-ON ring)
(ring TYPE solid)
(mowicoll GLUES ring)

Figure 9.3. An incomplete proof of Example 1.

The dotted lines from the above 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 ch-mem-assembly)

is an effect of the action

PRESS OBJECT ring ON ch-mem-assembly

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

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

are true in the current situation, that is

$$\big[(\text{mowicoll ISA adhesive}) \,\&\, (\text{mowicoll GLUES ring}) \,\&\, (\text{mowicoll GLUES ch-mem-assembly})\big] = \text{TRUE}$$

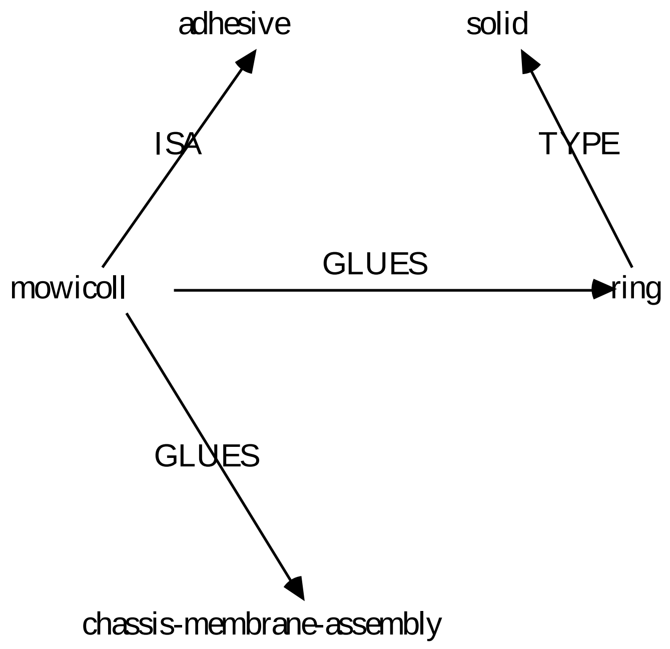
and the literal

(mowicoll BETWEEN ring ch-mem-assembly)

is not known to be true.

Comparing this proof tree with the one in figure 7.4, one may easily notice that it lacks some of the portions of the complete proof tree. Nevertheless, its leaves represent some important features of the objects from Example 1, features which, in the case of a weak theory, would correspond to the following explanation of Example 1:

Explanation 1:



(mowicoll ISA adhesive) & (mowicoll GLUES ring) &
(mowicoll GLUES chassis-membrane-assembly) & (ring TYPE solid)

Figure 9.4. The relevant features of Example 1, revealed by the proof tree in figure 9.3.

9.4 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 ch-mem-assembly)& (z ISA mowicoll)	*upper bound:* (z BETWEEN x y) *lower bound:* (z BETWEEN x y)& (x ISA ring) & (y ISA ch-mem-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 ch-mem-assembly)

taken from the premise of the inference rule

∀x ∀y ∀z [(z ISA adhesive) & (z GLUES x) & (z GLUES y) &
(z BETWEEN x y) ∅ (x 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 6.3, the system will also refine the model of the action 'PRESS'.

9.5 Generalization of the incomplete proof

Once the proof in figure 9.3 is built, the system will generalize it, as in a complete theory:

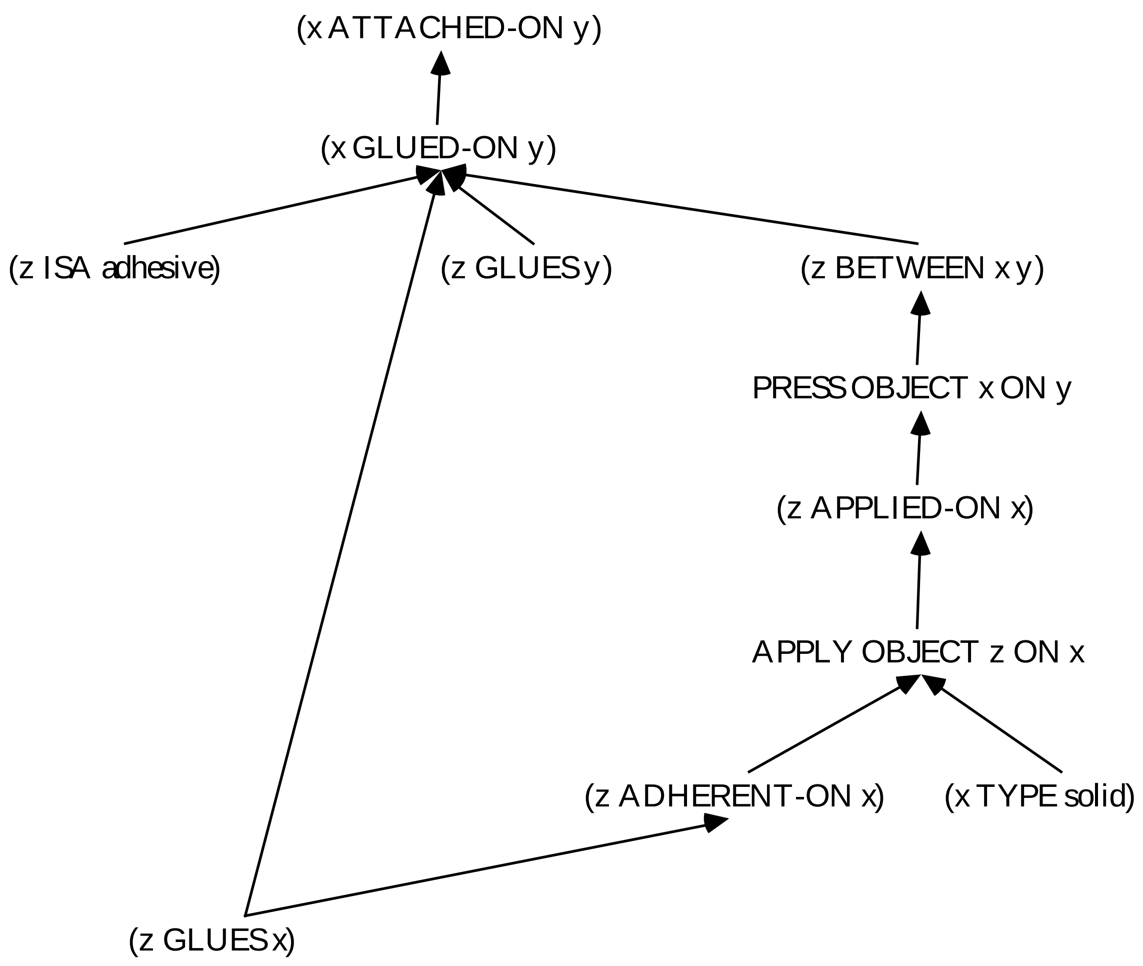


Figure 9.6. The generalization of the proof in figure 9.3.

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

9.6 Determining a reduced version space for the rule to be learned

As in the case of a weak theory, the Explanation 1 in figure 9.4 may be rewritten as a Lower Bound for the applicability condition of General Rule 1 (figure 6.3):

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

Also, the leaves of the generalized proof tree provide an **over-generalized explanation** of Example 1.

This over-generalized explanation corresponds to the **analogy criterion** from a weak theory and may therefore be considered as an Upper Bound for the applicability condition of General Rule 1 (see figure 6.3):

analogy criterion:

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

Figure 9.7. An over-generalization of Explanation 1 from figure 9.4.

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

IF

Upper Bound:

(x TYPE solid) &

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

Lower Bound:

(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 9.8. A reduced version space for the rule to be learned.

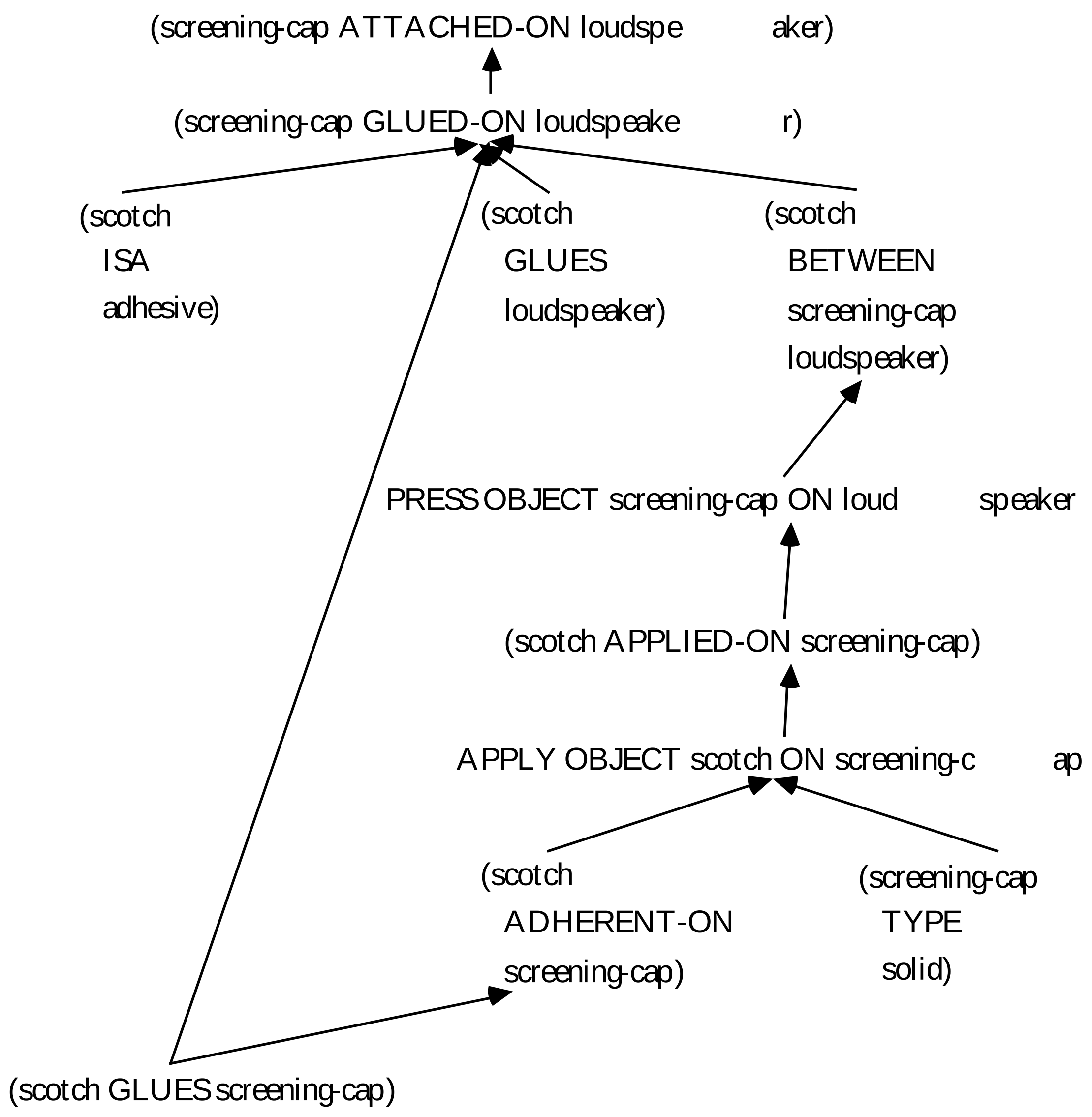
9.7 Searching the rule in the version space

As soon as the version space from figure 9.8 has been determined, rule learning will continue as in a weak theory. This time, however, the generalized proof tree in figure 9.6 provides a focus for the process of finding the explanations of the failures.

To illustrate this, let us consider again the failure in figure 8.12.

May I solve the problem
ATTACH OBJECT screening-cap ON loudspeaker
by solving the subproblems
APPLY OBJECT scotch ON screening-cap
PRESS OBJECT screening-cap ON loudspeaker ? <u>No</u>

In this case, the system generates the instance of the generalized proof in figure 9.6, corresponding to this problem solving episode (by replacing 'x', 'y', and 'z' with 'screening-cap', 'loudspeaker', and 'scotch', respectively).



(screening-cap ATTACHED-ON loudspeaker)
(screening-cap GLUED-ON loudspeaker)
(scotch ISA adhesive)
(scotch GLUES loudspeaker)
(scotch BETWEEN screening-cap loudspeaker)
PRESS OBJECT screening-cap ON loudspeaker
(scotch APPLIED-ON screening-cap)
APPLY OBJECT scotch ON screening-cap
(scotch ADHERENT-ON screening-cap)
(screening-cap TYPE solid)
(scotch GLUES screening-cap)

Figure 9.9. A wrong proof of the example from figure 8.12.

The fact that the user rejected the solution proposed by the system proves that the leaves of the instantiated tree in figure 9.9 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. 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.

In our case, the predicate

(scotch APPLIED-ON screening-cap)

is not true. Because this predicate should have been the effect of the action

APPLY OBJECT scotch ON screening-cap

it follows that the action 'APPLY' has a precondition which is not contained in its model and this precondition is not true in the current situation. This precondition is

(z TYPE fluid)

It is not satisfied in the case of the problem solving episode in figure 8.12 because 'scotch' (which is an adhesive tape) is not fluid. Therefore, the action 'APPLY' cannot be executed and its effect cannot be achieved.

It follows that the explanation of the failure in figure 8.12 is

NOT(scotch TYPE fluid).

Therefore, in an incomplete theory, finding the explanations of the failures reduces to finding the knowledge which is lacking from the knowledge pieces.

In this case, the generalized proof in figure 9.6 plays the role of a justification structure for the rule to be learned [Smith & al. 1985].

As it was also mentioned at the end of section 8.5.2, the generality of the learned rule is limited by the generality of the over-generalized explanation. 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 and some action models or inference rules from the associated generalized proof may be generalized to cover the new situation as well.

9.8 Learning in an imperfect theory

This section presents a future direction of our research: learning in an imperfect theory.

A theory may be imperfect from different points of view ([Mitchell & al. 1986; Rajamoney & DeJong, 1987]). However, we shall consider only imperfections resulting from the fact that certain pieces of knowledge (objects, inference rules, action models) are lacking from the domain theory or contain minor errors in their definitions in that parts of these definitions may be either more general or less general than they should be.

For instance, the preconditions of an action may be:

- more general than they should be, allowing the application of the action in situations in which the action is not applicable;
- less general than they should be, forbidding the application of the action in situations in which the action is applicable;
- may have some parts which are more general and other parts which are less general than they should be, having both of the above consequences.

The same types of errors may manifest in the effects of an action, in any of the two sides of an inference rule, or in the definition of an object property or relation.

Let us notice that the incomplete theory is a special case of the above defined imperfect theory. For instance, an action lacking some precondition predicates has a precondition that is more general than it should be. In particular, an action having no precondition is supposed to be applicable in any situation. Therefore, learning in an imperfect theory may be similar to learning in an incomplete theory.

Besides learning problem reduction rules, the system will also correct the theory. This will consist in appropriate generalizations and particularizations of parts of the knowledge pieces, in response to the encountered failures.

10. A SAMPLE SESSION WITH DISCIPLE IN THE TECHNIQUE DESIGN DOMAIN

10.1 General presentation

The design of technologies for the manufacturing of loudspeakers, at the Industrial Electronic Enterprise in Bucharest, has been chosen as a real world application for testing the problem solving and learning methods of DISCIPLE.

In this section we shall present in some detail this application, by commenting a sample session with DISCIPLE.

Our goal is to present the external behavior of the system.

The context for using DISCIPLE can be described by the interaction among the system, the human expert (i.e. the user of DISCIPLE), the background knowledge about the concerned technology (the domain theory which, in this case, is weak), and the current application of the system.

DISCIPLE and its user are in constant interaction, both proposing solutions and explanations to the other. The system has access to a knowledge base containing the domain theory, but its user does not.

The user has access to the features of the current application, but the system does not.

This last feature may be felt as a serious drawback. This is wrong because, while needing to know if the current loudspeaker has a certain feature, the system will simply ask the user, waiting for one of the following answers: yes, no, irrelevant, as exemplified in the next section.

DISCIPLE receives a very general specification of a problem (plan the manufacturing of a loudspeaker, in our example) and the system will start solving this problem without knowing the specific features of the current loudspeaker. Only when the known technological solution depends on the presence (or absence) of a certain feature, will the system ask the user if the loudspeaker has the respective feature.

The set of all questions asked during a session, together with their answers, provides a complete description of the current loudspeaker.

In this application DISCIPLE uses an implicit global control strategy which we consider suitable for the technology design domain in general.

The system will design the technology in three consecutive steps:

- design of the detailed (elementary) operations of the technology (i.e. use decomposition rules);
- design (choice) of the tools, devices or verifiers for each elementary operation (i.e. use specialization rules);
- design (choice) of the materials needed for each operation (i.e. use specialization rules).

During each of the above three stages, the system will implicitly use a depth first strategy in building the design tree. That is, it will always reduce the left most leaf of the tree, as long as there is a rule corresponding to the current stage. When there are no more nodes to reduce, the system will go to the next step, and the depth first strategy is resumed. Of course, the user may modify the system's global control. For instance, he may ask for the design of tools for a given action, although the system is in the first stage.

10.2 A sample design session

In this section we present a sample session with DISCIPLE. The session is put between /* ... */, our comments are outside these delimitors. Moreover, to differentiate between the user and the system, the answers of the first are underlined.

The session starts with the user indicating the top level action of manufacturing a loudspeaker. The system will present the possible reductions of the current operation (in this case MANUFACTURE), ranked in order of its preference. The user may accept the system's suggestion, may prefer a lower ranked reduction, if any, may reject all the system's suggestions, or may wish to modify the rule corresponding to one of the system's suggestions.

```
/*
    Solve the problem
     MANUFACTURE OBJECT LOUDSPEAKER.

    I propose the following solution
    MAKE OBJECT CHASSIS-ASSEMBLY.
    MAKE OBJECT MEMBRANE-ASSEMBLY.
    ASSEMBLE OBJECT CHASSIS-ASSEMBLY
                WITH   MEMBRANE-ASSEMBLY.
    FINISHING-OPERATIONS ON LOUDSPEAKER.

    Do you agree ? [Yes, No, Modify]: Yes

    To solve the problem
    MAKE OBJECT CHASSIS-ASSEMBLY.

    I propose the following solution
    FIX OBJECTS CONTACTS
        ON        CHASSIS.
    MAKE OBJECT MECHANICAL-CHASSIS-ASSEMBLY.
    FINISHING-OPERATIONS ON ENTREFER.

    Do you agree ? Yes
*/
```

DISCIPLE indicated a possible decomposition of the MANUFACTURE operation into simpler operations, and the user agreed with system's suggestion.

Further on, following the implicit control strategy, DISCIPLE proposed a decomposition of the action:

MAKE OBJECT CHASSIS-ASSEMBLY.

This decomposition was also accepted by the user.

```
/*

     LOUDSPEAKER HAS TERMINAL-BLOCK ? Yes
     To solve the problem
     FIX OBJECTS CONTACTS
          ON        CHASSIS.

     I propose the following solution
       *CRIMP OBJECTS CONTACTS
                ON          SUPPORT.
       *CONNECT OBJECT TERMINAL-BLOCK
                   ON       CHASSIS.
       Do you agree ? Yes
*/
```

DISCIPLE tried to decompose the FIX operation but the known rules are dependent on the loudspeaker features. Therefore, DISCIPLE asked the user a question and the answer received allowed it to choose the right rule.

It is by means of such questions that DISCIPLE becomes progressively aware of loudspeaker features. Describing the loudspeaker in advance would not only be too cumbersome, but even impractical, since we do not know in advance which features are important and which are not important for the technology design.

The system prints an asterisk before each action which may no longer be decomposed into simpler ones (CRIMP and CONNECT, in this case).

```
/*
       ...
     To solve the problem
      MAKE OBJECT MEMBRANE-ASSEMBLY.
```

I propose the following solution
CONNECT OBJECTS TERMINAL-WIRES
ON COIL.
ASSEMBLE OBJECT MEMBRANE
WITH COIL.
ASSEMBLE OBJECT MEMBRANE
WITH CENTERING-DEVICE.
FINISHING-OPERATIONS ON MEMBRANE-ASSEMBLY.
Do you agree ? *Yes*

To solve the problem
CONNECT OBJECTS TERMINAL-WIRES
ON COIL.

I propose the following solution
TIN OBJECTS TERMINAL-WIRES.
SHORTEN OBJECTS COIL-ENDS.
TIN OBJECTS COIL-ENDS.
TIN OBJECTS TERMINAL-WIRES
WITH COIL-ENDS.

Do you agree ? No

Could you indicate a solution ? *Yes*
SHORTEN OBJECTS COIL-ENDS.
SOLDER OBJECTS TERMINAL-WIRES
WITH COIL-ENDS.

Does your solution work because
COIL-ENDS PART-OF COIL ? *No*
COIL-ENDS ISA TERMINALS &
TERMINAL-WIRES ISA TERMINALS ? *No*

Could you give me an explanation ? *No*

```
    I cannot make any generalization
    I learned the following rule:

    Solve the problem
      CONNECT OBJECTS TERMINAL-WIRES
                ON       COIL.
    by solving the subproblems
      SHORTEN OBJECTS COIL-ENDS.
      SOLDER OBJECTS TERMINAL-WIRES
              WITH    COIL-ENDS.

    Do you agree ? Yes
*/
```

As stated in the previous section, the system has an implicit control strategy to choose the next problem to solve. Nevertheless, the user can direct the system to solve another problem. To this purpose it uses commands for traversing the problem-solving tree. Using such commands (not shown in our example) the user positioned DISCIPLE on the problem:

MAKE OBJECT MEMBRANE-ASSEMBLY.

Then, the implicit control strategy was resumed. Therefore, DISCIPLE proposed a decomposition of the MAKE operation. Then it automatically chose to decompose the first operation of this reduction (i.e. CONNECT) and proposed a possible decomposition of it.

The user rejected this decomposition because he had in mind another technological solution.

DISCIPLE tried to learn a general rule from user's solution. Therefore it started looking for an explanation of this solution. Because neither DISCIPLE nor the user were able to provide an explanation, no generalization was possible. Nevertheless, DISCIPLE retained the specific rule indicated by the user because it was told to do so.

This example shows that generalization is not always possible or desirable.

```
/*
    ...
   Unable to solve the problem
    ATTACH OBJECT RING
              ON      CHASSIS-MEMBRANE-ASSEMBLY.

   Could you indicate a solution ? Yes
   APPLY OBJECT MOWICOLL
         ON      RING.
   PRESS OBJECT RING
        ON      CHASSIS-MEMBRANE-ASSEMBLY.

   Does your solution work because:
   MOWICOLL GLUES RING ? Yes
   MOWICOLL GLUES CHASSIS-MEMBRANE-ASSEMBLY ? Yes
   RING PART-OF LOUDSPEAKER &
   CHASSIS-MEMBRANE-ASSEMBLY PART-OF LOUDSPEAKER ? No
*/
```

Since DISCIPLE did not know any reduction of the ATTACH operation, it asked the user for one and proposed an explanation of it, as shown in section 8.3.2.

Note that the explanation says why the decomposition is good and not why the ATTACH operation should be performed. Therefore the user's answer to the third question was "No".

```
/*
   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
```

```
May I solve the problem
ATTACH OBJECT SCREENING-CAP
          ON      LOUDSPEAKER.
by solving the subproblems
 APPLY OBJECT SCOTCH
         ON      SCREENING-CAP.
 PRESS OBJECT SCREENING-CAP
        ON      LOUDSPEAKER ? No

Could you give me an explanation ? Yes
NOT(SCOTCH TYPE FLUID)

I learned the following rule:
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
solve the problem
 ATTACH OBJECT x ON y.
by solving the subproblems
 APPLY OBJECT z ON x.
 PRESS OBJECT x ON y.
Do you agree ? Yes
```
*/

DISCIPLE proposed two other decompositions (consistent with the found explanation) which were validated by the user. This allowed the system to learn a more general rule, validated by the user.

During the second stage of technology design, the system will try to propose tools for the established operations, by using specialization rules.

/* ...

Unable to specialize the problem

CLEAN OBJECT ENTREFER.

Could you indicate a solution ? Yes

CLEAN OBJECT ENTREFER

WITH AIR-SUCKER.

Does your solution work because:

ENTREFER HAS DUST & AIR-SUCKER ABSORBS DUST ? *Yes*

This explanation may expressed as:

1. ENTREFER HAS DUST & AIR-SUCKER REMOVES DUST
2. ENTREFER HAS DUST & AIR-SUCKER GETS DUST

Choose the solution [number, No, Modify]: 1

*/

DISCIPLE was unable to propose any tool for the CLEAN operation, therefore it asked for a solution from the user. Then it found an explanation which was generalized.

/*

May I solve the problem

CLEAN OBJECT ENTREFER

by solving the specialization

CLEAN OBJECT ENTREFER

WITH AIR-PRESS ? *Yes*

May I solve the problem

CLEAN OBJECT MEMBRANE-ASSEMBLY

by solving the specialization

CLEAN OBJECT MEMBRANE-ASSEMBLY

WITH ACETONE ? *Yes*

May I solve the problem
CLEAN OBJECT MEMBRANE-ASSEMBLY
by solving the specialization
CLEAN OBJECT MEMBRANE-ASSEMBLY
        WITH   EMERY-PAPER ? *No*

Could you give me an explanation ? Yes
EMERY-PAPER DESTROYS MEMBRANE-ASSEMBLY.

I learned the following rule:
IF
Upper Bound:
(x HAS z) & (y REMOVES z) & NOT(y DESTROYS x)

Lower Bound:
(x HAS z) & (y ISA cleaner) & (y REMOVES z) &
(z ISA waste-material) & NOT(y DESTROYS x)
THEN
solve the problem
CLEAN OBJECT x
by solving the specialization
CLEAN OBJECT x WITH z.
Do you agree ? Yes
*/

DISCIPLE proposed other specializations of the CLEAN operation (analogous with user's solution) and synthesized a more general specialization rule.

/* ...
DRY OBJECT LOUDSPEAKER
    DEVICE DRYER.

Trying to constrain the dryer.
1. DECREASE COST.
2. DECREASE TIME.

```
    Choose the optimization criterion [Number, No]: 2

    I propose the following solutions
    1. TUNNEL-KILN
    2. CARROUSEL
    3. DRYING-SHELF

    Choose the solution: 1
*/
```

The system knows several specialized dryers. It also knows two meta-rules corresponding to the specializations of dryers: one for minimizing the cost and the other for minimizing the time.

The system is unable to choose among the dryers until the user expresses his preference for one of the above optimization criteria.

```
/*
    Print

    problem:
    MANUFACTURE OBJECT LOUDSPEAKER.

    loudspeaker's features:
    LOUDSPEAKER HAS TERMINAL-BLOCK

    optimization criterion:
    DECREASE TIME.

    partial technology:
    *CRIMP OBJECTS CONTACTS
            ON        SUPPORT.
    *CONNECT OBJECT TERMINAL-BLOCK
               ON      CHASSIS.
    ASSEMBLE OBJECT MECHANICAL-CHASSIS-ASSEMBLY.
    CLEAN OBJECT ENTREFER
           WITH   AIR-SUCKER.
    ...
```

```
     SHORTEN OBJECTS COIL-ENDS.
     SOLDER OBJECTS TERMINAL-WIRES
                WITH     COIL-ENDS.
     ASSEMBLE OBJECT MEMBRANE
                   WITH   COIL.
     ASSEMBLE OBJECT MEMBRANE
                    WITH   CENTERING-DEVICE.
     FINISHING-OPERATIONS OBJECT MEMBRANE-ASSEMBLY.
     ...
     *APPLY OBJECT MOWICOLL
                ON        RING.
     *PRESS OBJECT RING
               ON       CHASSIS-MEMBRANE-ASSEMBLY.
     ...
     *DRY OBJECT LOUDSPEAKER
             DEVICE TUNNEL-KILN.
       ...
*/
```

The user asked for a print-out of the technology so far designed.

The session continues until a detailed enough technology was designed.

The performance of DISCIPLE in the technique design domain is very encouraging. Presently, the knowledge base for the design of loudspeakers contains several hundreds of rules and objects.

The utility of DISCIPLE in technique design results from the fact that there are many types of products (belonging to a certain family as, for instance, the loudspeaker family) which are not very different from each other. As a consequence, many of the technological solutions used to manufacture a certain type of loudspeaker, are also applicable to a new type.

11. AN EXAMPLE FROM MANAGEMENT

In this section we shall present a hypothetical utilization of DISCIPLE in a management application which consists in defining the production and commercial strategy of an electronic company.

In this application, DISCIPLE acts as an aid to the manager of a company and, meanwhile, learns company management.

The theory of this domain is weak. It consists of knowledge about objects as, for instance, about electronic components (their features, who produces them and in what conditions, etc.), about possible partners, as well as knowledge about the possibilities of the manager's company (the components it can manufacture, general management strategies, etc).

In this domain, a problem for DISCIPLE is a management goal as the following one:

PENETRATE IN India WITH electronic-equipment

A solution to this goal consists of a plan of "elementary" actions which achieves it.

DISCIPLE has rules which decompose a problem into simpler problems or specialize a problem. These rules indicate in fact partial solutions to problems.

An example of a decomposition rule is the following one:

```
IF
    (x ISA new-product) &
    (y ISA market)
THEN
solve the problem
    PENETRATE IN y WITH x
by solving the subproblems
    GET OBJECT x
    MAKE-PUBLICITY TO x IN y
    SELL OBJECT x IN y WITH promotion-price
    SELL OBJECT x IN y WITH value-price
```

This rule gives a partial solution to the problem of penetrating a market with a new product.

Other rules would indicate how to make a good publicity or how to sell the product.

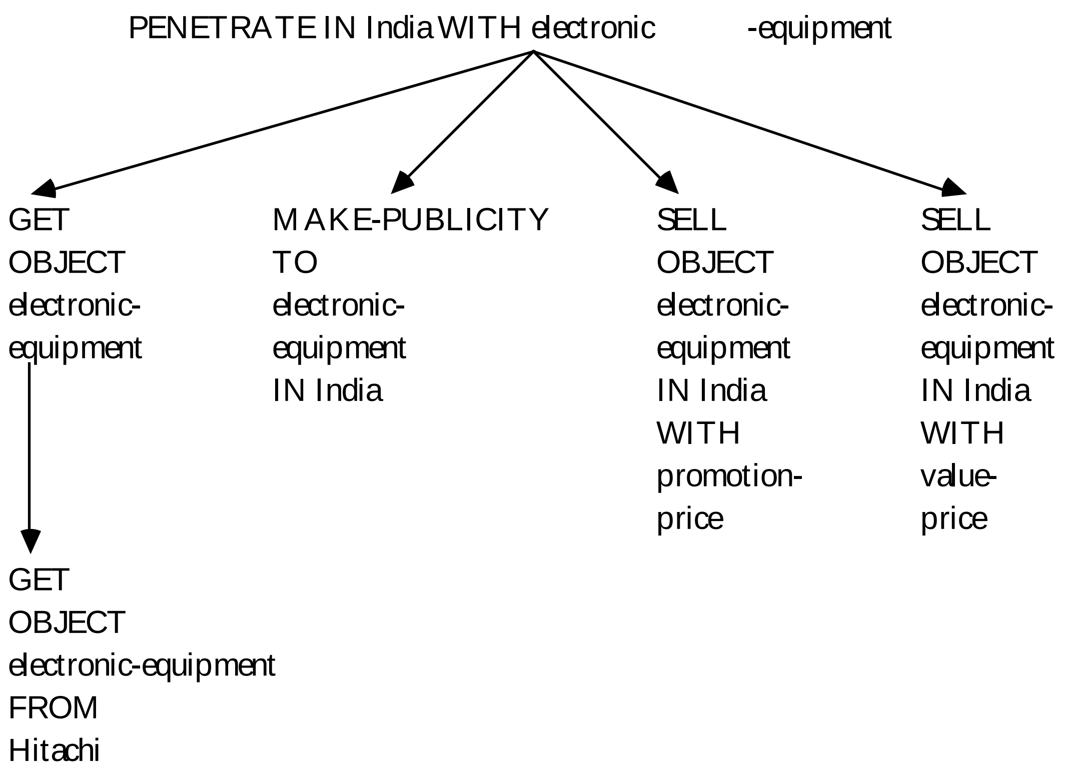
The following is an example of a specialization rule:

```
IF
    (x ISA product) &
    (y ISA company) &
    (y PRODUCES x)
THEN
solve the problem
    GET OBJECT x
by solving the specialization
    GET OBJECT x FROM y
```

The specialization rules are used to better define the problems to solve.

DISCIPLE solves a given problem by successively decomposing it into simpler subproblems and by specializing it to better defined problems. Therefore, the problem solving paradigm of DISCIPLE is problem-reduction.

During problem-solving, it develops a problem solving tree whose top represents the initial problem and whose leaves represent the solution to this problem:


PENETRATE IN India WITH electronic-equipment
GET OBJECT electronic-equipment
MAKE-PUBLICITY TO electronic-equipment IN India
SELL OBJECT electronic-equipment IN India WITH promotion-price
SELL OBJECT electronic-equipment IN India WITH value-price
GET OBJECT electronic-equipment FROM Hitachi

The leaves of the tree may be elementary actions (i.e. actions which can be directly executed) or general strategies which must themselves be reduced to elementary actions.

Let us suppose that, while trying to solve the problem of penetrating India with electronic equipment, the system encounters the following problem:

GET OBJECT vax-780

Suppose further that the user does not accept any of the system's suggestions ('BUY OBJECT vax-780 FROM Digital', for instance) and decide to make an exchange agreement with Digital, to buy 'vax-780' from it, and to sell 'displayscreen of type 0021' to it:

```
EXCHANGE-AGREEMENT WITH     Digital
                     TO-BUY  vax-780
                     TO-SELL displayscreen-0021
BUY OBJECT vax-780
     FROM    Digital
SELL OBJECT displayscreen-0021
      TO       Digital
```

Having received a solution from the user, the system is facing the problem of inferring a general rule, one instance of which is:

```
Solve the problem
        GET OBJECT vax-780
by solving the subproblems
        EXCHANGE-AGREEMENT WITH        Digital
                                       TO-BUY    vax-780
                                  TO-SELL  displayscreen-0021
        BUY OBJECT vax-780
              FROM   Digital
        SELL OBJECT displayscreen-0021
               TO         Digital
```

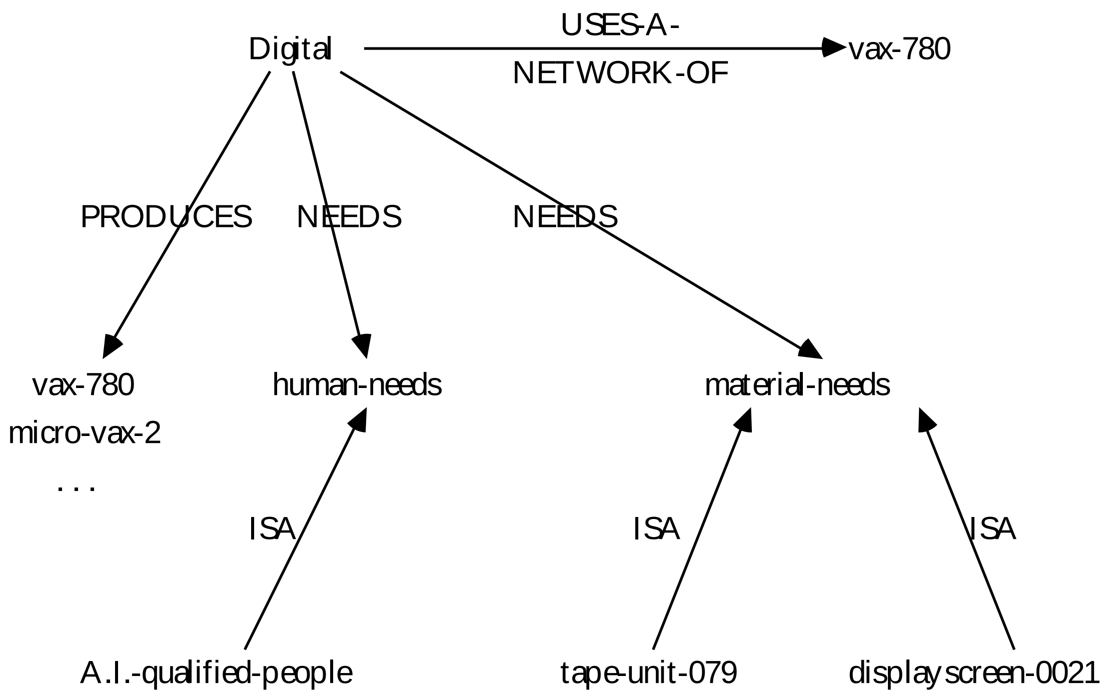
This instance suggests the system to learn a general rule of the form:

```
        IF
             v, d, ds satisfy <constraints>
        THEN
        solve the problem
             GET OBJECT v
        by solving the subproblems
             EXCHANGE-AGREEMENT WITH     d
                                        TO-BUY  v
                                        TO-SELL ds
             BUY OBJECT  v
                  FROM   d
             SELL OBJECT ds
                   TO      d
```

Note that 'EXCHANGE-AGREEMENT' might be an action previously unknown to the system.

Next, DISCIPLE will use its weak domain theory to find an explanation of user's solution.

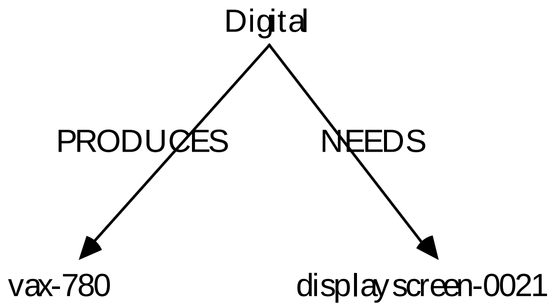
Let us suppose that the theory is represented by the following network:



DISCIPLE will look for an explanation in terms of the relations between the objects from the rule instance (vax-780, Digital, displayscreen-0021). It will propose partial explanations, asking the user to validate them:

Does your solution work because:
Digital PRODUCES vax-780 ? <u>Yes</u>
Digital USES-A-NETWORK-OF vax-780 ? <u>No</u>
Digital NEEDS displayscreen-0021 ? <u>Yes</u>

All the pieces of explanation validated by the user form the explanation of the example:

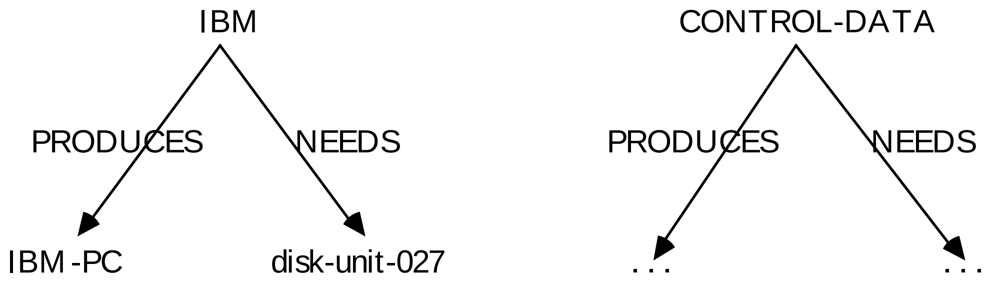


This explanation is used to define the following reduced version space for the rule to be learned:

```
IF
    Upper bound:
    (d PRODUCES v) &
    (d NEEDS ds)

    Lower Bound:
    (d ISA Digital) &
    (v ISA vax-780) &
    (ds ISA display-0021) &
    (d PRODUCES v) &
    (d NEEDS ds)
THEN
solve the problem
    GET OBJECT v
by solving the subproblems
    EXCHANGE-AGREEMENT WITH    d
                           TO-BUY  v
                           TO-SELL ds
    BUY OBJECT v
        FROM   d
    SELL OBJECT ds
         TO     d
```

Next, DISCIPLE will look in its knowledge base for other objects satisfying the upper bound of the above rule:



These objects will be used to generate instances of the above rule, the user being asked to validate or to reject them.

The positive examples thus produced will be used to generalize the lower bound of the above version space, and the negative examples will be used to particularize both bounds.

Finally, DISCIPLE will learn the following general rule:

```
IF
    (d ISA company) &
    (d PRODUCES v) &
    (d NEEDS ds) &
    (v ISA product) &
    (ds ISA my-product) &
    (ds STATE available)
THEN
solve the problem
    GET OBJECT v
by solving the subproblems
    EXCHANGE-AGREEMENT WITH     d
                              TO-BUY  v
                              TO-SELL ds
    BUY OBJECT v
        FROM   d
    SELL OBJECT ds
         TO      d
```

This rule may be expressed in English as follows: *"to get an object produced by a company needing one of my available products, make an exchange agreement, buy the object and sell my product".*

12. CONCLUSIONS

In this thesis we proposed an approach to the knowledge transfer from a human expert to an expert system. This approach is illustrated by DISCIPLE, an interactive system which integrates an empty expert system and a learning system, both using the same knowledge base.

With DISCIPLE, the building of a practical Expert System is a two-phase process.

In the first phase, the human expert has to introduce, into the knowledge base of DISCIPLE, elementary knowledge about the application domain. It is expected that this knowledge represents a "nonhomogeneous theory" of the domain, in that it provides complete descriptions of some parts of the domain, and incomplete or weak descriptions of other parts of the domain.

In the second phase, DISCIPLE is used as an interactive problem solver. From each contribution of the human expert to the problem solving process, the system is trying to learn the general problem solving rule illustrated by the user's solution. In this way, DISCIPLE progressively evolves from a helpful assistant in problem solving to a genuine expert.

With DISCIPLE, the critical process of building the "complete" knowledge base of an expert system is reduced to the process of building a smaller knowledge base containing only the theory of the application domain. Moreover, the resulting system is able to progressively improve its competence and performance in problem solving.

DISCIPLE integrates many learning and problem solving techniques. In spite of this, however, it appears as a unitary system, not only in what regards its external behavior, but also in what regards its internal behavior. The unity is given by the existence of a unique knowledge base which is organized around the notion of concept and supports the elementary operations with concepts (comparing the generality of concepts, generalizing concepts, and particularizing concepts).

The problem solving mechanisms of DISCIPLE consist in problem reduction, formulation, propagation and evaluation of constraints, and problem solving by analogy. These "classical" problem solving paradigms have been expressed in terms of the above mentioned elementary operations with concepts, and have been integrated into an advanced problem reduction method.

Trying to cope with the complexity of the real world applications, DISCIPLE makes the hypothesis that its theory about an application domain is nonhomogeneous, describing completely some parts of the domain, but only incompletely or even poorly, other parts of the same domain. This is a very difficult learning environment. However, DISCIPLE integrates different learning methods which allow it to learn at different levels of knowledge. A common feature of all these methods is that they are based on an understanding of the example from which the rule is learned.

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 important features of this learning method one could mention:

- the synergistic combination of different learning paradigm into a unitary learning method;
- 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 to 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. In this way, it is able to use a generalization of an incomplete proof of an example:

- for defining a justified analogy criterion;

- for finding the explanations of the failures;

- and as a justification structure of the general rule to be learned.

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 (clever questions to the user, use of analogy, etc.). It also opens a new research direction: learning in an imperfect theory, a generalization of the incomplete theory.

Another important effect of learning in the context of a weak theory or an incomplete theory is that of developing the domain theory.

Let us 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 the 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.

We have implemented DISCIPLE in Le_Lisp [Chailloux, 1985] and we have used it to design techniques for the manufacturing of loudspeakers.

There are several **weaknesses** of DISCIPLE, on which will shall direct our future research.

For instance, the generality of the learned rule is limited by the generality of the over-generalized 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 and some action models or inference rules from the associated generalized proof 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 meta-rules for finding far off explanations;

While DISCIPLE uses control knowledge in the form of meta-rules, such knowledge is not learned, having to 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 meta-explanations that can become meta-preconditions on the use of the rules.

An important future direction of research consists in developing the learning methods of DISCIPLE in order to be able to deal with an imperfect theory in which the knowledge may contain minor errors.

There are also several **lessons** we have learned from the design of DISCIPLE.

One is that, *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 weak theories is short-time harmful.* Indeed, forcing the expert to completely formalize a domain theory (which may even not have such a complete theory) may result in a degradation of the knowledge provided by him/her.

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

APPENDIX

In this appendix we present other learning methods which are complementary to DISCIPLE, and might therefore be used to further develop our theory and methodology of expert knowledge acquisition. More precisely, we show:

- how one might improve the quality and efficiency of the empirical generalizations.

- how one might learn hierarchies of concepts (which constitute elementary knowledge in DISCIPLE);

LEARNING BASED ON CONCEPTUAL DISTANCE

Abstract

We present a new approach to concept learning from examples and concept learning by observation, which is based on a intuitive notion of conceptual distance between examples (concepts) and combines symbolical and numerical methods. Our approach is supported by the observation that very different examples generalize to an expression that is very far from each of them, while identical examples generalize to themselves. Therefore, a generalization of two examples, as well as the process of obtaining this generalization, represents indications of the conceptual distance between the examples. Following this idea we propose some domain independent and intuitively justified estimates for the conceptual distance. Usually however, a set of examples may be characterized by several generalizations, each suggesting a certain conceptual distance. The minimum of these is taken as the estimation of the real conceptual distance. Moreover, the corresponding generalization is recommended as the one to be made by the learning system because this generalization has the desirable property of reflecting the greatest number of common features of the examples. We also present a hierarchical conceptual clustering algorithm which groups objects so that to maximize the cohesiveness (a reciprocal of the conceptual distance) of the clusters. We further show that conceptual clustering may improve learning from complex examples describing objects and the relations between them. The idea is that learning good generalizations of such examples requires matching the most similar objects which, in turn, requires a clustering of these objects. Finally we present a methodology of learning hierarchies of prototype objects which is a step towards automating the construction of knowledge bases for expert systems.

Keywords: Learning from examples, Preferable generalization, Conceptual distance, Concept learning by observation, Conceptual cohesiveness, Conceptual clustering, Hierarchies of prototype objects.

I. INTRODUCTION

Machine learning may be defined as any process by which a computer increases its knowledge and improves its skills.

One of the basic types of learning is inductive learning, that is, learning by generalizing specific facts or situations.

Inductive learning has received considerable attention in Artificial Intelligence ([Cohen & Feigenbaum, 1982], [Dietterich & Michalski, 1981], [Langley, 1987], [Michalski, Carbonell & Mitchell, 1983, 1986], [Mitchell, Carbonell & Michalski, 1985]). Two different kinds of inductive learning are: learning from examples and conceptual clustering.

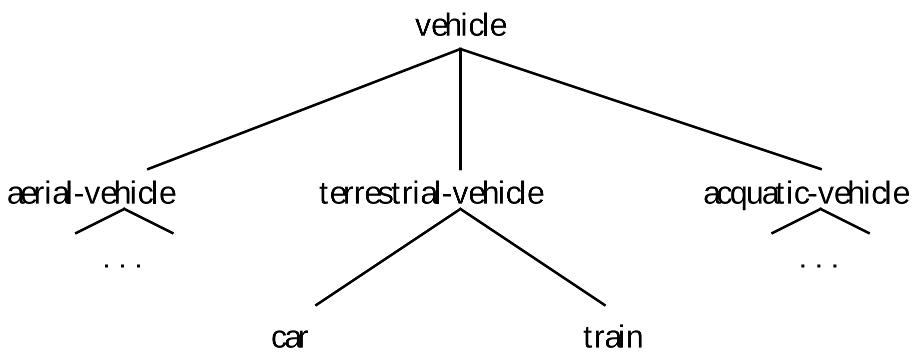
In *concept learning from examples*, the learning system is presented with independent instances representing a certain class, and the task is to induce a general description of the class. The instances can be specific physical objects, actions, processes, images, etc. Let us suppose that they are different cars (CITROEN, RENAULT, OPEL, etc). In this case, the system's task is to learn the concept of car, represented by what is common to all the given examples (objects with four wheels, used to transport people, etc). Having formed such a concept, the system will be able to recognize other objects as being or not being cars, as they have or have not the properties of the car concept.

In *conceptual clustering*, the learner is also presented with a set of examples, but these examples are no longer said to represent the same class. In this case, the learner has to solve two problems:

- the *aggregation problem* of distinguishing classes (defined as extensionally enumerated sets of objects) into which the examples can be grouped;

- the *characterization problem* of inducing an intentional description for each class.

The examples presented to the learner could be, for instance, descriptions of specific cars, ships, airplanes, or trains, and the system would learn the following concepts:



As defined, the characterization problem is very similar with the problem of learning from examples. Conceptual clustering processes must address this problem since the quality of a clustering is dependent on the description of the clusters (the simplicity of these descriptions, the map between these descriptions and the clusters they cover, etc. [Michalski & Stepp, 1983a]).

Although nobody is claiming that the aggregation and characterization problems should be independent, the present conceptual clustering algorithms [Fisher & Langley, 1985] first solve the aggregation problem, and then use the methods of learning from examples to obtain a description for each cluster. The so obtained descriptions are further used to estimate the quality of the clustering and may suggest to search for another clustering.

In this paper we present a new approach to concept learning from examples and concept learning by observation, which is based on a intuitive notion of *conceptual distance* [Michalski & Stepp, 1983a] between examples (concepts) and combines symbolical and numerical methods.

Our approach is supported by the observation that very different examples generalize to an expression that is very far from each of them, while identical examples generalize to themselves. Therefore, a generalization of two examples, as well as the process of obtaining this generalization, represent indications of the conceptual distance between the examples. Following this idea, in section III, we propose some domain independent and intuitively justified measures for the conceptual distance.

However, the process of obtaining a generalization of a set of examples is not a deterministic one. Several generalizations are possible, and each suggests a certain conceptual distance between them. Therefore, we propose to estimate the real distance by the minimum of these distances. Moreover, the corresponding generalization is recommended as the one to be made by the learning system since this generalization has the desirable property of reflecting the most commonalties between the examples.

In section IV, we present a conceptual clustering algorithm which is based on the reciprocal of the conceptual distance, called *conceptual cohesiveness* [Michalski & Stepp, 1983a], and on a partial ordering defined on conceptual cohesiveness. The main feature of the conceptual cohesiveness is that it takes into consideration not only the properties of the individual objects, but also their relationship to other objects and, most importantly, their relationship to some pre-defined concepts characterizing object collections.

To cluster a set of examples E1, ... En, our algorithm first looks for the two examples Ei, Ej for which the conceptual cohesiveness is maximum. These examples form the seed of a cluster. A new example Ek is added to this cluster only if the conceptual cohesiveness of the set {Ei,Ej} is not greater than the conceptual cohesiveness of the set {Ei,Ej,Ek}.

While, in general, only conceptual clustering is based on learning from examples, in our approach learning from *complex* examples is also based on conceptual clustering. Here by a complex example we mean an example describing several objects and the relations between them. The idea is that learning good generalizations of complex examples requires matching the most similar objects which, in turn, requires a clustering of these objects. This method is presented in section V. For instance, it should be of use in Scene Analysis where the recognition of each individual scene component and the recognition of the whole scene are dependent of each other.

We consider that the approach presented in this appendix is also significant to automatic knowledge acquisition for expert systems and, in section VI, we present a methodology for generating hierarchies of prototype objects.

II. CONCEPT LEARNING FROM EXAMPLES

Concept learning from examples means forming a general description (concept) of a class of objects given a set of objects (examples) from this class.

We assume that both the examples and the concepts are described in the same representation language, as conjunctions of literals. For instance, Fig. 1 represents two examples of toy trains. They are taken from the famous Michalski's train problem [Michalski, 1984], which consists in finding a common characterization of a set of such trains.

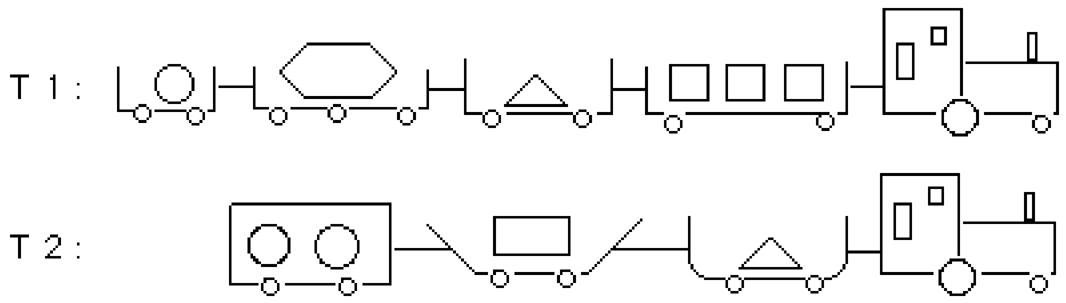
As can be seen in Fig. 1, each train is described as a conjunction of literals of the form (p a1,...,an), where p is a predicate and a1,...,an are the arguments of the predicate. For instance:

(car-shape open-rectng C2)

means that the shape of the car C2 is an open rectangle, and

(length long C2)

means that the length of C2 is long.



T1:
(infront C1 C2) (infront C2 C3) (infront C3 C4) (infront C4 C5)
(length long C1) (length long C2) (length short C3) (length long C4)
(length short C5) (car-shape machine C1)(car-shape open-rectng C2)
(car-shape sloping-top C3) (car-shape open-rectng C4)
(car-shape open-rectng C5) (contains C2 L2) (contains C3 L3)
(contains C4 L4) (contains C5 L5) (load-shape square L2)
(load-shape triangle L3) (load-shape hexagon L4)
(load-shape circle L5) (nrpts-load 3 L2) (nrpts-load 1 L3)
(nrpts-load 1 L4) (nrpts-load 1 L5) (nr-wheels 2 C1)
(nr-wheels 2 C2) (nr-wheels 2 C3) (nr-wheels 3 C4)(nr-wheels 2 C5)

T2:
(infront C6 C7) (infront C7 C8) (infront C8 C9)
(length long C6) (length short C7) (length short C8) (length short C9)
(car-shape machine C6) (car-shape U-shape C7)
(car-shape open-trapeze C8) (car-shape closed-rectng C9)
(contains C7 L7) (contains C8 L8) (contains C9 L9)
(load-shape triangle L7) (load-shape rectangle L8)
(load-shape circle L9) (nrpts-load 1 L7) (nrpts-load 1 L8)
(nrpts-load 2 L9) (nr-wheels 2 C6)(nr-wheels 2 C7)(nr-wheels 2 C8)
(nr-wheels 2 C9)

Figure 1. The first two trains from Michalski's train problem

The argument of a predicate, also called term, may be a constant, a variable, or f(t1,...,tn), where f is a function and t1,...,tn are terms.

For each variable a domain is defined, containing all possible values the variable can take. As in [Michalski & Stepp, 1983a], we distinguish among nominal (categorical), linear (quantitative), and structured variables, whose domains are unordered, totally-ordered, and graph-oriented sets, respectively. Structured variables represent generalization hierarchies of related values as, for instance, the following one:

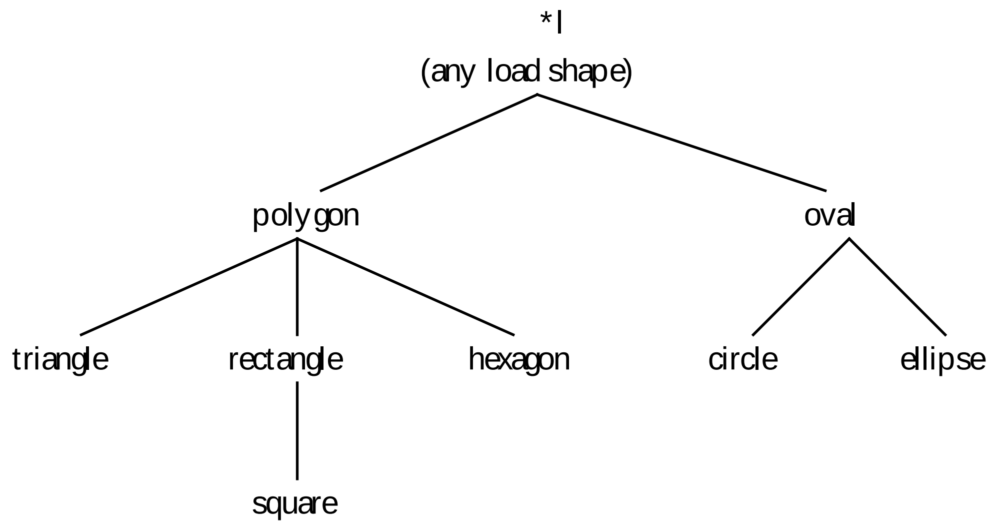


Fig. 2. A generalization hierarchy for the load shapes in Fig.1.

The predicates may be related by theorems as, for instance, the following one:

∀x∀y∀z, (contains x y) & (contains y z) ––> (contains x z)

We shall use the predicates from Michalski's train problem to present our approach. The following, for instance, are two simple examples:

E1: (car-shape open-rectng A1) (length short A1)
E2: (car-shape open-trapeze A2) (length short A2)

Fig. 3. Two simple examples.

The first example describes the car A1 as having the shape open rectangle and being short. The second example describes the car A2 as having the shape open trapeze and being also short.

A *generalization* of an example is an expression which "describes" a set containing the example. That is, by replacing the variables of the generalization by suitable constants, one finds back the example (see precise definition below).

For instance, the following is a generalization of E1:

G: (car-shape open-rectng x) (length y x)

It describes a set of open rectangle cars of any length. One finds back E1 by replacing 'x' by 'A1' and 'y' by 'short'.

A *generalization* of several examples E1,...,En is an expression which describes a set containing all these example. For instance, the following is a generalization of E1 and E2:

G1: (car-shape z x) (length short x)

It describes a set of short cars of any shape.

A key characteristic of the concept learning from example problem is that there is an important structure inherent to the language used to represent the concepts. This structure is based on the relation **less-general-than**, which was defined in the section 3.7 of the thesis.

For instance

E1: (car-shape open-rectng C1)

is less general than

G: (car-shape z x) (length y x)

Indeed, one may use the theorem that any car has a length

$\forall v\ \exists u$ (length u v) = TRUE

and may rewrite E1 as

E1: (car-shape open-rectng C1) (length u C1)

Now, there exists the substitution σ = (z<–open-rectng, x<–C1, y<–u) such that 'σoG=E1'. Therefore, E1 is less general than G.

Let us consider again the examples in Fig. 3. Some of their generalizations are the following ones:

G1:	(car-shape z x) (length y x)
G2:	(car-shape z x) (length short x)
G3:	(car-shape z x) (length y t)
G4:	(car-shape x y)
G5:	(length short x)
G6:	(length x y)

A learning system will always have the problem of choosing among the competing generalizations. We define the notion of *preferable* generalization as being the generalization which is less general than all the other generalizations.

In our case, the *preferable* generalization is G2 since it is less general than all the other generalizations. For instance, G2 is less general than G1 because there is the substitution 'σ = (y <– short)' such that σoG1=G2.

The notion of *preferable* generalization is relative to the knowledge about the learning universe and cannot be considered absolute. New knowledge may lead to an improvement. For instance, let us consider that we have acquired new knowledge about car shapes. Suppose that this knowledge is expressed by the following hierarchy:

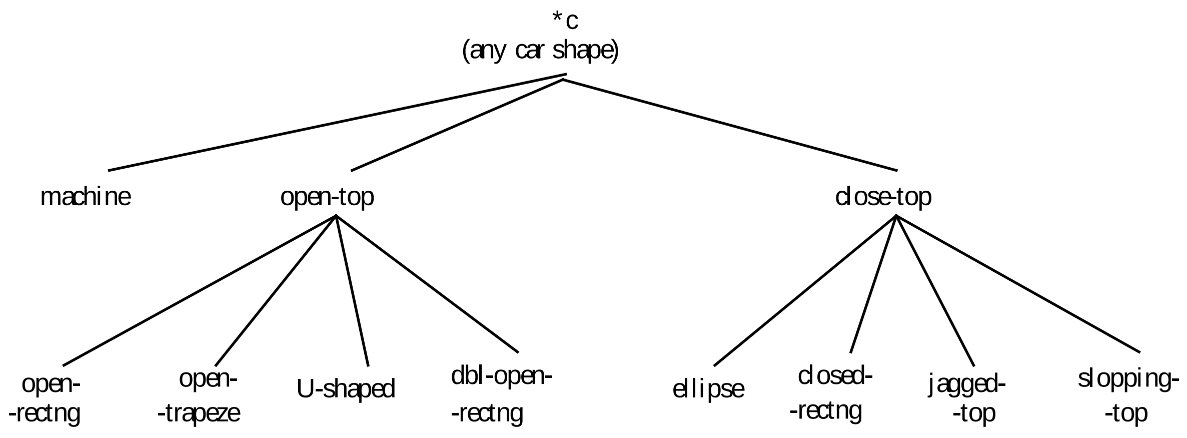


Fig. 4. A generalization hierarchy for the shapes of the cars in Fig. 1.

In this case

G7: (car-shape open-top x) (length short x)

is also a generalization of E1 and E2. Since G7 is less general than G2 (there is 'σ =(z<–open-top)' such that 'σoG2=G8'), G7 is less general than all the other generalizations. Therefore G7 is the preferable generalization in the new context.

Given a set of generalizations, it is most probable that there is no generalization that is the least general. For instance, the set {G1, G3, G4, G5, G6} does not contain a least general expression.

Given two expressions G1 and G2, if neither G1 *is less general than* G2 nor G2 *less general than* G1, then G1 and G2 are said to be *incomparable* from a generalization point of view.

In a given learning situation, there are many generalizations which are incomparable, and the main difficulty of learning is to choose the right one [Kodratoff & al. 1984]. Therefore, we have to look for another, more relaxed, definition of the *preferable* generalization.

Let us notice that, if Gi is the *preferable* among G1, ... ,Gn, then the set of instances of Gi is included into the set of instances of each of these generalizations. It follows that the number of instances of Gi is less than the number of instances of any other generalization. Based on this observation we may compare two generalizations G1 and G2 even if they are incomparable from a generalization point of view. We shall say that G1 is preferable if the number of instances of G1 is less than the number of instances of G2.

But this definition is still unsatisfactory for the simple reason that learning is not an isolated aim. One is learning in a given universe, with a well-defined goal in that universe. As a consequence, a generalization has also to point to the essential common properties of the examples. Therefore, a *good* generalization is not one which represents many of the properties common to the examples, but that one which represents many of the *important* properties common to the examples.

Let us consider, for instance, the case of a robot which learns concepts from examples representing physical objects. Each object is described by specifying the actions which could be performed on it, the relations which could be established between this object and other objects, the shape of the object, its color, etc. Although all these properties are relevant for a robot their relative importance depends on robot goals. If the robot intends to use the learned concepts for action planning, then the action and relation properties are to be considered more important the the color, for instance. On the other hand, if the robot intends to use the learned concepts for recognizing objects, then the color property should be considered more important.

This problem is treated in detail in section VI.

In our approach, the relative importance of the predicates must be defined by the teacher. One way to do it is to associate a weight to each predicate. Based on the weights of the predicates we could estimate the relevance (value) of a generalization as the sum of the weights of the predicates included into the generalization. We may therefore consider that the *preferable* generalization is the one which maximizes the relevance and minimizes the number of instances.

We may now informally define the *preferable* generalization of the examples E1, E2, ... , En, as follows:

- if there is a generalization Gi which is less-general-than all the other generalizations of the examples then Gi is the *preferable* generalization;

- if Gi1, ... , Gim, are all the incomparable generalizations of the examples, then consider the *preferable* generalization the one which is the *most relevant* (contains the most important predicates) and has the *least number of instances*.

In the following section we shall propose a more computational definition of the *preferable* generalization, definition based on domain-independent and intuitively justified heuristics.

III. CONCEPTUAL DISTANCE AND CONCEPTUAL COHESIVENESS

A. The general approach

Given two descriptions, we could notice similarities and dissimilarities. For instance, the descriptions:

E1: (car-shape open-rectng B1)

E2: (car-shape U-shaped B2)

are similar because both are characterized by the predicate *car-shape* and each car-shape is *open-top* (see Fig. 4), but the first shape is *open-rectng*, while the second one is *U-shaped*.

If we could estimate the similarity S(E1,E2) and the dissimilarity D(E1,E2) between the descriptions E1 and E2, then we could estimate the conceptual distance between E1 and E2 by a function of S and D. This function would quantify the contribution of S and D to the conceptual distance.

Since a generalization of two examples is able to reveal subtile commonalties between these examples it seems to be a suitable means of estimating their conceptual distance. Indeed, let us notice that very different examples generalize to a *general* expression that is very far from each of them, while identical examples generalize to themselves. Moreover, we want to take into account that the fewer changes are made to the examples in order to obtain a generalization, the greater the similarities and the less the dissimilarities are.

Here we shall propose an estimation of the conceptual distance which is based on the learning algorithm developed at LRI [Kodratoff & Ganascia, 1986]. This algorithm uses the principle of structural matching: the examples are successively transformed until they acquire approximately the same form. Then the generalization is obtained by retaining only the common features.

To illustrate this algorithm let us consider the following two examples:

E1: (car-shape open-rectng C1) (contains C1 L1)
(car-shape open-trapeze C3)

E2: (car-shape U-shaped C2) (length short C2)

The first example represents two cars, an open rectangle one containing an object and an open trapeze one. The second example represents a short U-shaped car (see Fig. 4).

The algorithm first rewrites the examples revealing their common features:

E1: (car-shape open-top X1) (contains X1 L1)
(car-shape open-trapeze C3)
(X1<–C1)

E2: (car-shape open-top X1) (length short X1)
(X1<–C2)

Next, it will use the theorems of the representation language in order to reveal in one example features exhibited by the other. Such a theorem expresses, for instance, the fact that any object has a length:

$\forall z\ \exists t$ (length t z)=TRUE

Using this theorem one rewrites the two examples as follows:

E1: (car-shape open-top X1) (length Y1 X1) (contains X1 L1)
(car-shape open-trapeze C3)
(X1<–C1, Y1<–t)

E2: (car-shape open-top X1) (length Y1 X1)
(X1<–C2, Y1<–short)

If one example exhibits a certain feature more times than the other, one uses the idempotency of the AND operator to make the feature appear the same number of times:

(car-shape open-top X1) =
(car-shape open-top X1) & (car-shape open-top X1)

Therefore, the two expressions are farther rewritten as follows:

E1: (car-shape open-top X1) (length Y1 X1) (contains X1 L1)
(car-shape open-top X2)
(X1<–C1, Y1<–t, X2<–C3)

E2: (car-shape open-top X1) (length Y1 X1)
(car-shape open-top X2)
(X1<–C2, Y1<–short, X2<–C2)

When no other common features may be revealed, one simply drops the differences between the two expressions and obtains the generalization of the initial examples:

G(E1,E2): (car-shape open-top X1) (length Y1 X1)
(car-shape open-top X2)
(may-be-the-same X1 X2)

Let us notice that the operations made in order to obtain a structural matching and a generalization of the examples are indications of similarities and dissimilarities between these examples.

The predicates of E1, E2, and G(E1,E2) could be classified in four categories (thus obtaining four lists of predicates), as follows:

COMMON
Predicates from G(E1,E2) which were initially present in E1 and E2:
{ (car-shape open-top X1) }

THEOREMS
Predicates introduced in G(E1,E2) by using the theorems of the representation language:
{ (length Y1 X1) }
IDEMPOTENCY

Predicates introduced in G(E1,E2) by using the idempotency of the **AND** operator:

{ (car-shape open-top X2) }

DROPPED

Predicates dropped from E1 and E2, in order to obtain G(E1,E2):

{ (contains X1 L1) }

The general intuition is that each such type has a specific influence to the estimation of the similarities and dissimilarities between the examples.

In the following sections we shall propose and justify measures for each type of predicates.

B. Common predicates

Let us consider the following four examples:

E1: (car-shape open-rectng C1)

E2: (car-shape U-shaped C2)

E3: (car-shape open-rectng C3)

E4: (car-shape ellipse C4)

Since each of these four descriptions is characterized by the same predicate, the conceptual distance between them is exclusively determined by the distance between their arguments.

Intuitively, distance(E1,E3) < distance(E1,E2) < distance(E1,E4)

Let us also consider the following generalizations (see Fig. 4):

G(E1,E3): (car-shape open-rectng X1)

G(E1,E2): (car-shape open-top X2)

G(E1,E4): (car-shape *c X3)

Let us notice that *open-rectng, open-top,* and **c* are all values from the structured domain in Fig. 4, and that *open-rectng* is less general than *open-top* which in its turn is less general than **c*. While *open-*

rectng is an instance, *open-top* is a generalization with 4 instances and **c* is a generalization with 9 instances.

We could define the *degree of generality* of an argument, as the ratio of the number of argument's instances to the total number of instances from argument's domain, that is:

$$g(a) = \frac{\text{number of instances of "a"}}{\text{number of instances of the domain of "a"}}$$

For example:

g(open-rectng) = 0.   g(open-top) = 0.44   g(*c) = 1.

This definition applies also to the so-called linear (quantitative) variables [Michalski & Stepp, 1983a].

Let us notice that there is no dissimilarity between E1 and E3. Indeed, C1 and C3 are just different names for the same entity (i.e. the car) and X1, in the expression *G(E1,E3)=(car-shape open-rectng X1)*, is just another name for the car. X1 denotes a definite object and, therefore, g(X1)=0.

Intuitively, the more similar two descriptions containing only common predicates are, the less general are the arguments of their generalizations. Also, the more dissimilar two such descriptions are, the more general are these arguments.

All the arguments of a predicate being a priori of the same importance, one should propose an estimation for the similarity (dissimilarity), between two examples E1 and E2, by a function of the mean degree of generality of the arguments of the generalization G(E1,E2).

Let us now consider the following three examples:

E5: (COLOR RED C5)

E6: (COLOR BLUE C6)

E7: (SIZE BIG C7)

and the generalization:

G(E5,E6): (COLOR *d X8)

It is quite obvious that E5 is more similar to E6 than to E7, in spite of the fact that the arguments of G are variables. The simple fact that E5 and E6 are characterized by the same predicate makes them similar. Therefore, the similarity estimation function should also indicate a certain similarity between two examples even when all the arguments of G are variables, but both examples are characterized by the same predicate.

To sum up, let us consider two examples

E1: (P a1 a2 ... an) & . . .

E2: (P b1 b2 ... bn) & . . .

and their generalization

G(E1,E2): (P c1 c2 ... cn) & . . .

Let also g(ci) be the generality degree of the argument ci.

Then we propose to estimate the contribution of P to the dissimilarity and similarity between E1 and E2 by the following functions on the degree of generality of the arguments of P:

$D(E1,E2,P) = 0.5(\Sigma g(ci)) / n$

$S(E1,E2,P) = 1 - D(E1,E2,P)$

That is, we take the total contribution of a predicate P to the conceptual distance between E1 and E2 as being equal to 1, and we distribute it between similarity and dissimilarity in accordance with the generality degree of its arguments.

C. Dropped predicates

Let us consider two examples and their generalization:

E1: (car-shape open-rectng C1) & (contains C1 L1)

E2: (car-shape U-shaped C2)

G(E1,E2): (car-shape open-top X1)

In order to obtain a generalization of E1 and E2 one has to drop the predicate *contains* because it represents a feature of E1 which has nothing in common with any feature of E2. Therefore, this predicate is an indication of dissimilarity between the two examples.

Therefore we propose to estimate the contribution of a dropped predicate P to the estimation of the dissimilarity and similarity between E1 and E2 as follows:

D(E1,E2,P) = 1

S(E1,E2,P) = 1 - D(E1,E2,P) = 0

D. Predicates introduced by idempotence

We consider that the necessity of using idempotency of the logical 'AND' (for computing a generalization of two descriptions) is an indication of dissimilarity. But this dissimilarity has to be considered less than in the case of dropping predicates. Indeed, the predicate involved is present in both descriptions (that is, both descriptions have the property expressed by the predicate) but a different number of times.

Let us consider the following examples:

E1 = (length short C1)

E2 = (length short C2)(length short C3)

E3 = (length short C4)(length long C5)

We could obtain the following generalizations (by applying idempotency in the first example):

G(E1,E2) = (length short X1)(length short X2)

G(E1,E3) = (length short X3)(length   *l X4)

Intuitively, one sees that distance(E1,E2) < distance(E1,E3).

G(E1,E2) and G(E1,E3) differ only by the predicate which was introduced by idempotency. The only significant dissimilarity between (length short X2) and (length *l X4) consists in the generality degree of the first argument: g(short)=0, g(*l)=1.

As in the case of the common predicates, we could estimate the dissimilarity, due to a predicate introduced by idempotency, by the mean of the degree of generality of its arguments.

For intuitive, but also for formal reasons, we cannot accept any contribution of idempotency to the similarity estimation . Indeed, if idempotency would contribute to similarity estimation, then the similarity of two descriptions would be undefined and arbitrary since idempotency can be applied any number of times.

Therefore, the contribution to the estimation of the dissimilarity and similarity between E1 and E2 of the predicate P, introduced in G(E1,E2) by idempotency, is taken as follows:

$D(E1,E2,P) = 0.5(\Sigma g(ci)) / n$
$S(E1,E2,P) = 0.$

The generalization algorithm will always prefer idempotency to dropping, but will also try to use idempotency as few times as possible.

E. Predicates introduced by theorems

In principle, these predicates should be treated as the common predicates. Let us consider, for instance, the examples:

E1: (on A B)
E2: (near C D)

We may use the theorem '∀x∀y (on x y) ∅ (near x y)' and rewrite the first example as:

E1: (on A B) (near A B)

In this way we have revealed a common feature of E1 and E2: both represent two objects which are *near* one another.

However, we must take care avoiding counting several times the contribution of a predicate to the similarity and dissimilarity estimation, as shown by the following example.

Let us consider, for instance, the examples

E1: (on A B)

E2: (on C D)

and their rewritten form

E1: (on A B) (near A B)

E2: (on C D) (near C D)

In this case we should consider the *on* predicate as the only predicate *common* to the examples. Adding *near* would almost mean counting several times the *on* predicate.

F. Relative importance of the predicates

A system learning concepts from examples is supposed to have its own goals that are intended to be achieved by using the learned concepts. Since these goals are also known by the teacher supplying the examples, it is reasonable to suppose that the examples specify only the properties relevant to the system's goals. Even in such a case however some properties may be regarded as more important than others.

We assume that the relative importance of the properties is given by the teacher in the form of a weight associated to each predicate. Therefore, the previously estimated contributions of a predicate, to the similarity and the dissimilarity of two descriptions will be multiplied by the weight of the predicate. For instance, if P is a common predicate with the weight 'w', then its contribution to the dissimilarity and similarity estimation will be taken as:

$$D(E1,E2,P) = 0.5w((\Sigma g(ci)) / n$$

$$S(E1,E2,P) = 1 - D(E1,E2,P)$$

G. Estimation of conceptual distance and conceptual cohesiveness

Let us consider two examples E1, E2, and one of their generalizations G(E1,E2). As shown in the previous sections, one may estimate the contribution to similarity and dissimilarity of each involved predicate. By adding these estimations, one obtains the total estimation of similarity S(E1,E2,G) and the total estimation of dissimilarity D(E1,E2,G). These estimations depend of course on the generalization G(E1,E2). Another generalization G'(E1,E2) would produce different estimations D'(E1,E2,G') and S'(E1,E2,G').

Having estimated the similarity and the dissimilarity between E1 and E2 (corresponding to G(E1,E2)), one is able to estimate the conceptual distance between E1 and E2 (corresponding to G), as a function of D and S. Hereafter we shall consider the following distance function:

$$f(E1,E2,G) = D(E1,E2,G)/S(E1,E2,G)$$

Using G'(E1,E2) instead of G(E1,E2), one obtains another estimation of the conceptual distance between E1 and E2:

$$f(E1,E2,G') = D(E1,E2,G')/S(E1,E2,G')$$

We take, as the conceptual distance between E1 and E2, the minimum of f(E1,E2,G) over all possible generalizations of E1 and E2:

$$\text{conceptual-distance}(E1,E2) = \underset{G(E1,E2)}{\mathit{MIN}} \{f(E1,E2,G(E1,E2))\}$$

Moreover, *the generalization for which f is minimum is recommended as the concept to be learned* from E1 and E2 since this generalization has the desirable property of revealing the greatest number of common features between the examples and of being the least general among the generalizations revealing the same amount of common features.

Since the definition of the conceptual distance is based on the generalizations of the examples, this definition applies for any number of examples.

Let us consider n examples E1,E2,...,En and G(E1,E2,...,En), one of their generalization. Based on this generalization one can compute the four lists:

CM: predicates from G(E1,...,En) which were initially present in E1,...,En;

TH: predicates from G(E1,...,En) which were introduced in at least one example, by using the theorems of the representation language (but not idempotency);

ID: predicates from G(E1,...,En) which were introduced in at least one example by using the idempotency of the **AND** operator;

DR: predicates dropped from E1,...,En, in order to obtain G(E1,...,En).

Further on, using the above four lists, one can estimate the similarity S(E1,...,En,G), the dissimilarity D(E1,...,En,G), and the conceptual distance f(E1,...,En,G). The minimum of f taken over all possible generalizations of E1,...,En is the conceptual distance between E1,...,En:

$$\text{conceptual-distance}(E1,...,En) = \mathop{MIN}_{G(E1,...,En)} f(E1,...,En,G)$$

The reciprocal of the conceptual distance is called the *conceptual cohesiveness* of the set {E1,...,En}. The more similar and less dissimilar are the examples, the greater is their conceptual cohesiveness.

In the next section we shall present a hierarchical clustering algorithm based on the above notion of conceptual cohesiveness.

IV. CLUSTERING BY GENERALIZING

Conceptual clustering was introduced by Michalski and Stepp [Michalski & Stepp, 1983a] as an extension of processes of numerical taxonomy (a collection of methods used to form classification schemes over data sets) [Van Ryzin, 1977]. The main quality of the conceptual clustering is that it is able to capture the "Gestalt properties" of object clusters, that it, properties that characterize a cluster as a whole and are not derivable from properties of individual entities.

To illustrate this idea, let us consider the following example taken from [Michalski & Stepp, 1983a]:

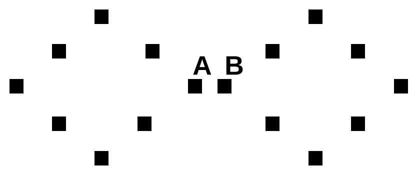


Fig. 5. An illustration of conceptual clustering.

A person considering this figure would typically describe the observed points as representing two diamonds. Thus, the points A and B, although closer to each other than to other points, are placed into different clusters.

CLUSTER/2 [Michalski & Stepp, 1983a] is a conceptual clustering algorithm able to make such classifications. In this section we present another clustering algorithm which is based on the conceptual cohesiveness defined in the previous section.

The goal of our clustering algorithm is to group the examples in such a way so that to maximize the conceptual cohesiveness of the clusters.

Our algorithm is based on the following observation: if {E1,E2,...,En} is a cluster with *high* conceptual cohesiveness and {Ei, ... , Ek} is a subset of this cluster, then the conceptual cohesiveness of {Ei, ... , Ek} is a good approximation of the conceptual cohesiveness of {E1,E2,...,En}. Let us now suppose that we add a new example Ea to this cluster. If the conceptual cohesiveness does not decrease significantly, then Ea belongs to the same concept as {E1,E2,...,En}. Otherwise, {E1,E2,...,En} and Ea belong to different concepts.

To illustrate this idea, let us consider that each example represents either a man or a woman. The conceptual cohesiveness of each subset of women is approximately the same with the conceptual cohesiveness of the set of all women. However, adding a man to such a set would significantly decrease the conceptual cohesiveness of the set.

To make this approach operational, one has to be able to determine when a conceptual cohesiveness decreases significantly. This is analogous to the definition of what we call *resolution*, a measure that indicates the minimum distance that has to exist between two elements to be perceived as distinct.

Our claim is that, in a given domain, one could experimentally determine the resolution by measuring the distances between concepts thought as distinct.

We propose to express this resolution as a threshold $\rceil\mu$, where $0<\rceil\mu\leq 1$, and to state that the conceptual cohesiveness of two sets S1 and S2 are different if and only if

cohesiveness(S1) is less than μ*cohesiveness(S2)

or

cohesiveness(S2) is less than μ*cohesiveness(S1).

Let {E1, ... , Et} be the examples to be clustered. The clustering algorithm first determines the pair of examples {Ep,Eq} for which the conceptual cohesiveness is maximum and takes it as the *seed* of a cluster. A new example is introduced into this cluster only if it does not decrease the cohesiveness of the cluster below the cohesiveness of {Ep,Eq}.

Once a cluster is completed, it replaces, in the set of examples, all the examples it contains, and the process is restarted with the new examples.

In greater detail the clustering algorithm is the following one:

Step 1: ask for the resolution of the application domain.

The resolution of the application domain is defined by the user as a threshold μ, where $0<\rceil\mu\leq 1$.

Given two sets S1 and S2, cohesiveness(S1) < cohesiveness(S2) if and only if cohesiveness(S1) is less than μ*cohesiveness(S2).

If neither "c(S1) < c(S2)" nor "c(S2) < c(S1)" we say that c(S1) and c(S2) are incomparable, were by c(Si) we denoted the cohesiveness of Si.

Step 2: compute the conceptual cohesiveness of each pair of examples.

Let E = {E1, ... ,En} be the set of examples.

For each pair {Ep,Eq} compute its cohesiveness by using the generalizations G(Ep,Eq) and the corresponding quadruples

L(Ep,Eq) = (CO(Ep,Eq), TH(Ep,Eq), ID(Ep,Eq), DR(Ep,Eq))
as presented in section III.

Step 3: choose a seed of the clustering.

Determine the pair {Ep,Eq} for which c(Ep,Eq) is maximum. If several such pairs exist, choose one of them.

Let {Ep,Eq} be the chosen pair. It is called *seed* of the clustering.

G(Ep,Eq) is one of the *most* relevant concepts among those represented by pairs of examples.

Let M = {Ep,Eq}.

We shall discover a first cluster by introducing in M other elements from E.

Step 4: determine the examples which could be members of the cluster represented by the chosen seed.

That is, determine the set:

T = { Ek | "c(Ep,Eq) and c(Ek,Ep) are incomparable"
"c(Ep,Eq) and c(Ek,Eq) are incomparable" }

Let us suppose that c(Ek,Ep) < c(Ep,Eq). In this case also c(Ep,Eq,Ek) < c(Ep,Eq) so Ek may not be member of the concept represented by {Ep,Eq}.

Step 5: introduce examples into the cluster.

For each Ek from T, if c(Ep,Eq) and c(M,Ek) are incomparable, then introduce Ek into M.

Initially M = {Ep,Eq}. Therefore, c(M,Ek) means c(Ep,Eq,Ek).

If M = {Ep,Eq, ... ,Et} then c(M,Ek) means c(Ep,Eq, ... ,Et,Ek).

At the end of this step one has discovered the cluster represented by the seed {Ep,Eq}:

M = {Ep,Eq, ... ,Es}.

Step 6: replace the examples contained in the cluster with the cluster.

Remove from the set of examples E, the elements of the discovered cluster M = {Ep,Eq, ... ,Es}.

Consider M as a *complex* example Em and introduce it into E.

That is, E ♦ (E - M) U {Em}.

If Ei is an initial example and Em = {Ep,Eq, ... ,Es}, is a *complex* example, then we consider that

G(Ei,Em) = G(Ei,Ep,Eq, ... ,Es) and

c(Ei,Em) = c(Ei,Ep,Eq, ... ,Es)

If Ms is a cluster containing the *complex* example Em then Ms represents a super-concept of the concept Em.

Step 7: rerun the algorithm.

Repeat from step 1 with the new set of examples until E is reduced to one element (Ek) or to two elements (Ek1,Ek2). Ek (respectively G(Ek1,Ek2)) is the concept representing all the examples.

The presented algorithm is only a basic one. Several improvements are obvious. For instance, let us consider again step 2. If E = {Ei1,...,Eit} U {Em}, Em being the last *complex* example formed, then we have to consider only the pairs {Eik,Em} since the other pairs have been already considered in the previous steps 1.

One could also modify the step 3 of the above algorithm by working with several seeds simultaneously. Indeed, instead of choosing one seed (among the incomparable ones) and to determine the corresponding cluster, one could consider all the competing seeds at the same time and determine simultaneously the corresponding clusters.

Let us also notice that although the discovered clusters are disjoint with respect to the clustered examples (or one is included into the other) their descriptions are not guaranteed to be disjoint. That is, they may have common instances.

Since an example of using this algorithm is presented in section 6, here we will only make clear its differences with CLUSTER/2 [Michalski & Stepp, 1983a]. Both these algorithms are able to discover hierarchies of concepts and each concept is represented as a conjunction of predicates. Even more, they are both based on a notion of seed. However, in CLUSTER/2 a seed is an example, while in our algorithm a seed is represented by a pair of examples.

CLUSTER/2 requires as input the number of clusters to be determined. The analog in our algorithm is the "resolution". CLUSTER/2 may successively consider different numbers, until it finds the right one. Similarly, our algorithm may do experiences with different resolutions.

Another difference between these algorithms is that CLUSTER/2 is looking for non-overlapping concepts that optimize pre-defined criteria, while our algorithm is looking for the most relevant concepts (as defined in section III), be these overlapping or not.

Finally, we may contrast the presented algorithm with the other clustering algorithm developed in our team [Benamou & Kodratoff, 1986]. While both algorithms combines symbolical and numerical methods, the presented algorithm is more symbolically oriented while the other one relies more on the numerical approach. The two approaches complement each other in a natural way. While the clustering algorithm presented in this paper tends to discover more relevant concepts, it also requires more processing resources.

V. GENERALIZING BY CLUSTERING

In this section we present an interesting relation which exists between generalization and clustering, in our approach.

Recall, from the previous section, that our clustering algorithm uses generalizations in order to cluster. We shall show that computing *good* generalizations of complex examples requires in turn a clustering phase.

By complex examples we mean examples containing objects. For instance, the examples in Fig. 1 contain car and load objects. The description of an object "O", from such an example, consists of all the predicates containing "O" as an argument. For instance, the description of the first car of the first train is the following one:

C1: (car-shape machine C1) (length long C1) (infront C1 C2)
(nr-wheels 2 C1)

The description of the load in the second car is:

L2: (contains C2 L2) (load-shape square L2) (nrpts-load 3 L2)

One may easily notice that the description of a train is the conjunct of the descriptions of the objects it contains, except that, in the latter, some predicates are duplicated. For instance, (infront C1 C2) appears both in the description of C1 and in that of C2.

Provided that the two trains in Fig. 1 are examples of a general train concept, the objects from these examples are instances of the objects from the general train description. Therefore, one way to determine the general train concept is to match and generalize the descriptions of the objects from the examples.

Our claim is that for *computing good generalizations of complex examples one should match the most similar objects.*

We shall illustrate this generalization strategy by applying it to compute a generalization of the two trains in Fig. 1.

First of all we extract from each of the two train examples the descriptions of the objects. We find 9 objects (5 cars and 4 loads) in the first example and 7 objects (4 cars and 3 loads) in the second one.

Next we look for the two objects (one from E1 and the other from E2) for which the conceptual distance is minimum. These objects will be matched, that is, they are supposed to represent the same object in the generalization of E1 and E2. This matching will of course influence the following matchings. For instance, if two cars Ci and Cj are matched, and are represented in the generalization by X1, then the loads contained into these cars (Li and respectively Lj) became more similar to each other. This is represented by replacing, in the descriptions of Li and Lj, the predicates (contains Ci Li) and (contains Cj Lj) by (contains X1 Li) and (contains X1 Lj), respectively.

Having established the matching between the most similar objects, we look for the other two objects which are the most similar. We continue this way, matching each object from E1 with an object from E2 (using idempotency, if needed).

We find the generalization of E1 and E2 as the union of the generalizations of the corresponding objects (eliminating of course the identical predicates):

(car-shape machine X1)(length long X1)(infront X1 X5)(nr-wheels 2 X1)
(car-shape * X2)(length short X2)(contains X2 Y1)(infront X2 X4)
(infront * X2)(nr-wheels 2 X2)(load-shape triangle Y1)(nrpts-load 1 Y1)
(car-shape * X3)(length short X3)(contains X3 L2)(infront X4 X3)
(nr-wheels 2 X3)(load-shape circle   Y2)(nrpts-load * Y2)
(car-shape open-top X4)(length * X4)(contains X4 Y3)(nr-wheels * X4)
(load-shape polygon Y3)(nrpts-load 1 Y3)
(car-shape open-top X5)(length * X5)(contains X5 Y4)(infront X5 *)
(nr-wheels 2 X5)(load-shape polygon Y4)(nrpts-load * Y4)

This description represents a type of train having the following features:

- the first car is a two wheel machine. It is followed by a two wheel open car containing polygons;

- the last car is a two wheel short one, containing circles. It is preceded by an open car containing one polygon which, in its turn, is proceeded by a two wheel short car containing a triangle;

When there are more than two examples to generalize, one needs to cluster the objects and to match objects belonging to the same cluster.

The main advantage of this approach is that it reduces a process of finding a generalization of complex descriptions to several processes of finding generalizations of much simpler descriptions.

The clustering of the objects may raise, however, complex combinatorial problems. In such a case, one may use heuristics (or a very simple clustering algorithm) to limit the objects to be clustered. For instance, it should be easy to establish that one must try to match cars with cars and loads with loads. Moreover, one does not need to cluster all the examples, but only to find out one cluster containing an object from each example.

VI. ACQUIRING OBJECT KNOWLEDGE

In this section we shall illustrate the relevance of our learning approach to the automation of knowledge base construction for expert systems.

A commonly used method of representing knowledge in artificial intelligence systems is to use prototypes ([Bobrow & Winograd, 1977], [Kodratoff & Tecuci, 1987a], [Minsky, 1975], [Sridharan & Bresina, 1983]). Each prototype represents a class of objects which is relevant for the system's application domain. The prototype is a parameterized representation of the properties common to the objects in the class. These prototypes are ordered in a class-subclass hierarchy in which each prototype inherits the properties of its super-class prototypes. The main feature of such a representation is that knowledge is organized around conceptual entities, in a memory efficient manner.

Therefore, automated construction of hierarchies of prototypes is an attempt towards automated construction of knowledge bases for expert systems.

The clustering algorithm presented in section IV is able to discover a hierarchy of concepts characterizing a set of examples. The only thing which remains to be done, for building a hierarchy of prototypes, is to fill up this structure, by computing a description for each concept (node in the tree). Each such description has to be in terms of its ancestors in the tree, inheriting and particularizing their descriptions.

We shall illustrate this problem in the robot world [Tecuci & al. 1983] presented in Fig. 6. It consists of mechanical parts to be used in assembling tasks.

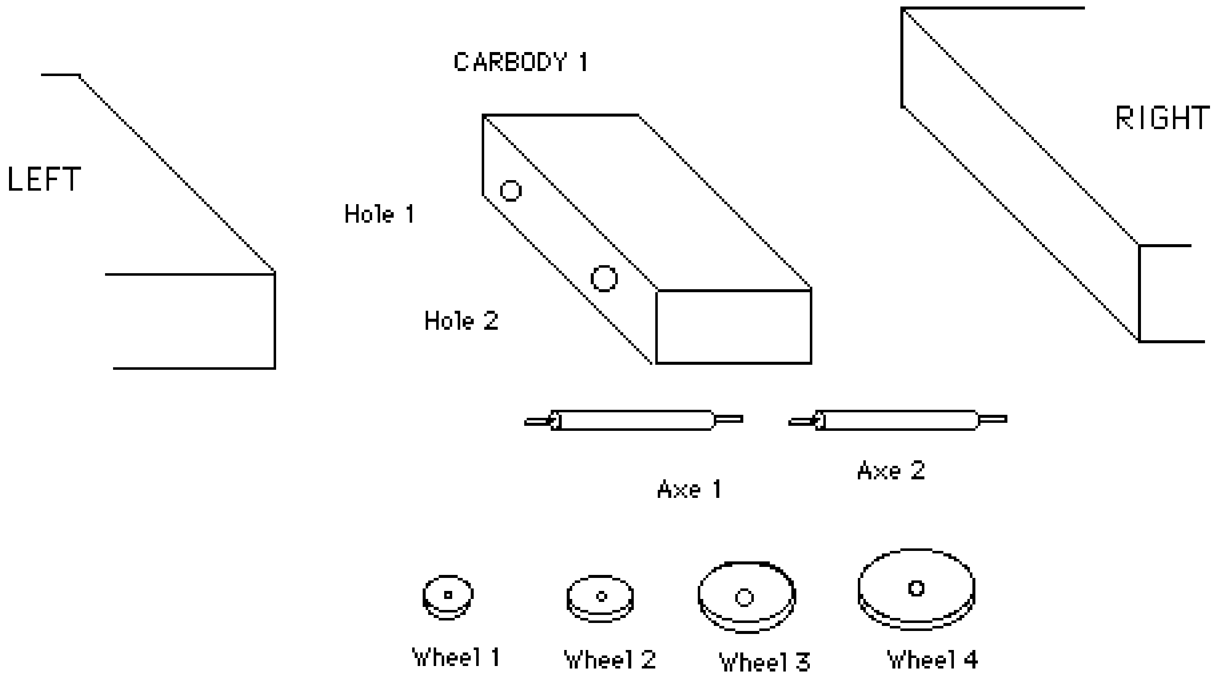


Fig. 6. A robot assembly world.

The robot is told the description of each part (AXLE1, AXLE2, WHEEL1, WHEEL2, ...) and is asked to learn the general concepts represented by these examples. Such concepts are, for instance, axle, wheel, graspable-object etc. The goal is to use the learned concepts in planning assembling tasks (for instance, planning the assembly of a car).

For instance, the following is the description of AXLE1:

(RELATION ATTACHED R1) (RELATION ATTACHED R3)
(RELATION THRU R2) (ACTION GRASP R2) (ACTION MOVE R2)
(ACTION INSERT R1) (ACTION INSERT R3) (POSITION P2)
(GRASPING G2) (APPROACHING L2)
(SUBPART SOLID CYLINDER(5 1) R1)
(SUBPART SOLID CYLINDER(20 4) R2)
(SUBPART SOLID CYLINDER(5 1) R3) (ALIGNED R1 R2 R3)

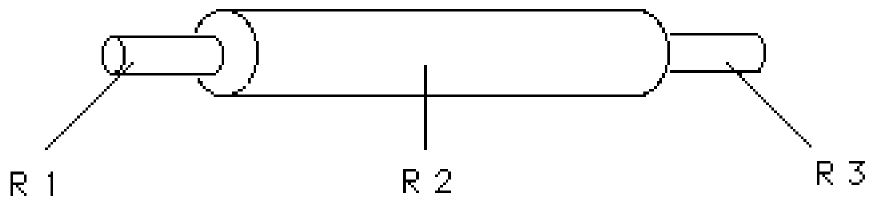


Figure 6'. Details of an axle.

The symbols R1, R2, and R3 are the names of AXLE1's sub-parts. P2, G2, L2, are constants representing spatial positions. R1 and R3 could be in the relation ATTACHED with other entities (ATTACHED to a WHEEL, for instance) and R2 could be in the relation THRU with other entities (THRU a CARBODY HOLE, for instance). The actions which could be performed on AXLE1 are GRASP (by grasping R2), MOVE (by moving the grasped sub-part) and INSERT (by inserting R1). AXLE1 is also characterized by a position (P2), a grasping point (G2) and a corresponding approaching point (L2). The three sub-parts R1, R2, R3, are solid cylinders, two of them (R1 and R3) having the same dimensions (height and diameter). The parts are aligned.

The robot is also given that the predicates describing the possible relations between parts or the actions that may be performed with these parts are to be considered more important than the predicates describing the shape of the parts. This information is given by the teacher in the form of the following weights associated with the predicates:

w(RELATION) = w(ACTION) = 4
w(GRASPING) = w(APPROACHING) = w(POSITION) = 3
w(SUBPART) = 2
w(ALIGNED) = w(INSIDE) = w(PARALLEL) = 1

Running our clustering algorithm with these examples we obtained the following hierarchy of concepts:

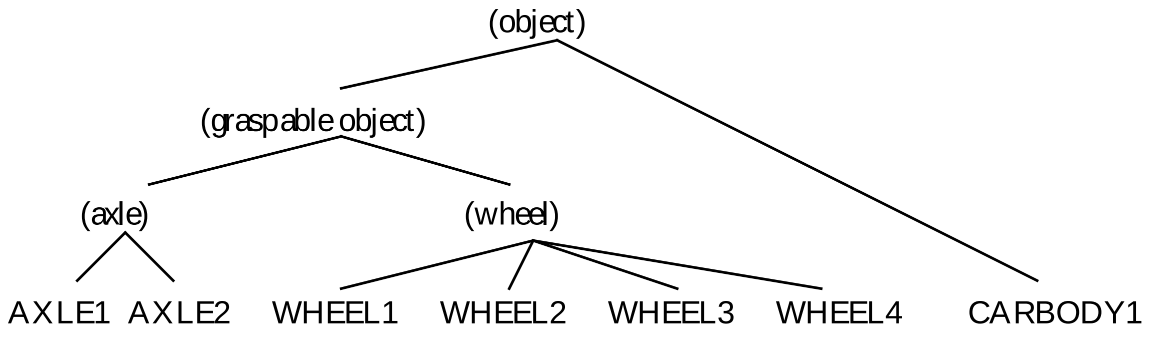


Fig. 7. Concepts learned from the robot world in Fig. 6.

Notice that the robot discovered the concepts that are relevant to its goal (planning assembling tasks).

Let us now suppose that the robot goal is to recognize objects. In this case, the teacher will have to state that the most important predicates are those describing the shape of the parts and the robot may discover the following concepts:

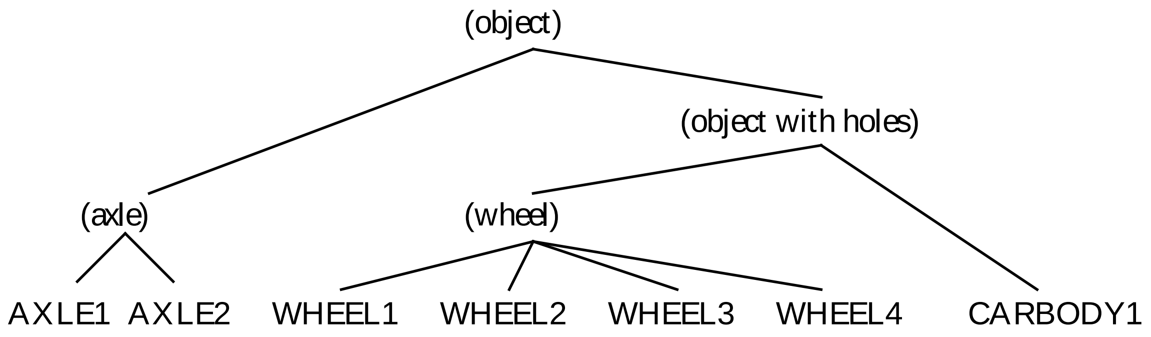


Fig. 8. Another set of concepts learnable from Fig. 6.

Let us consider again the hierarchy in Fig. 7. We want to transform it into a hierarchy of prototypes [Tecuci 1984a]. In such a hierarchy, each prototype is defined in terms of its ancestors. For instance, one says that *graspable object* is an object that has specific properties and that *axle* is a *graspable object* that has specific properties. The description that is actually associated with a prototype consists of its specific properties. Therefore, the description of *object* consists of the features common to all the objects from Fig. 6. Also, the description of *graspable object* consists of the features common to the axles and wheels, except those that are already present in the description of *object* because they are automatically inherited.

Our clustering algorithm has already computed a description of each concept. Therefore, to transform the hierarchy in Fig. 7 into a hierarchy of prototypes one has only to remove, from the description of each concept, the features which are already present into the descriptions of its ancestors. Acting this way one obtains the following hierarchy of prototypes:

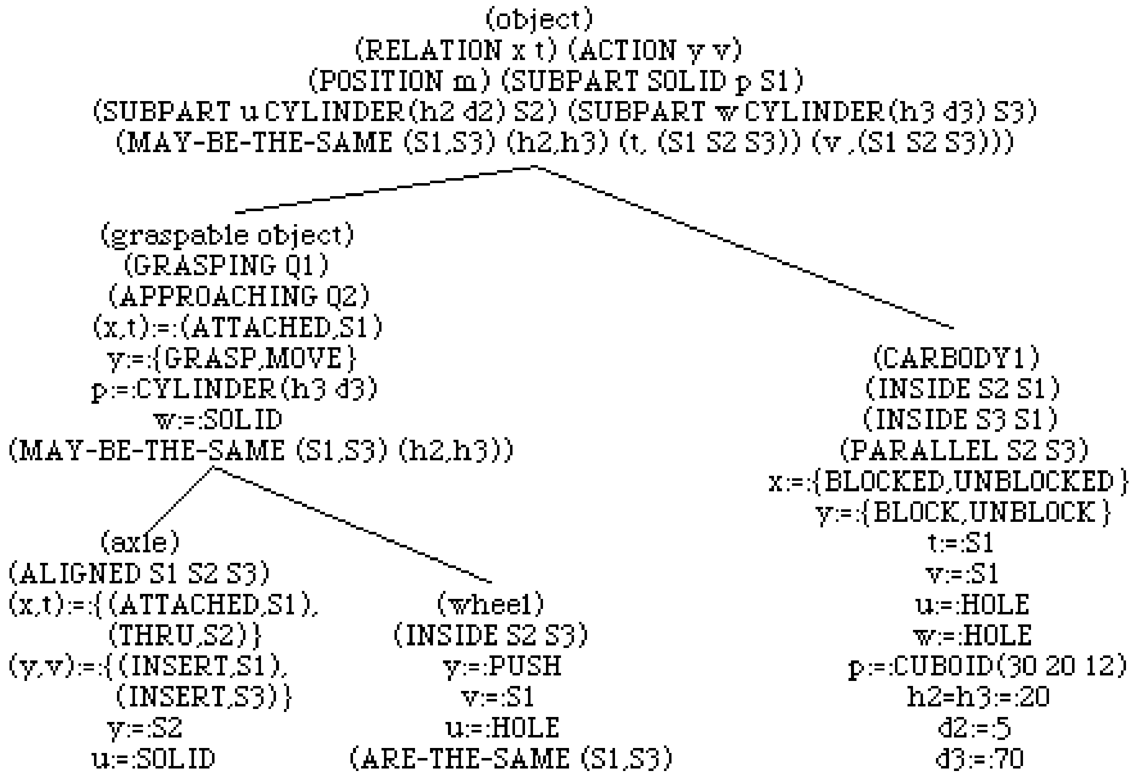


Fig.9. A hierarchy of prototypes learned from examples.

The prototype *object* contains the properties common to all objects in the robot world presented. These properties express the fact that an object (be it a graspable one or not) could be in certain relations with other objects (there is no relation common to all objects and this is the reason for the presence of the variables in the description), that certain actions can be performed on the object, that the object is characterized by a certain spatial position and has cylinder sub-parts of the same height. Implicitly, different names means different entities. Therefore the MAY-BE-THE-SAME predicate indicates which variables can take the same value. For instance "v" could take the same value as S1 or S2 or S3.

The prototype *graspable object* contains the properties common only to graspable objects (has a GRASPING point and a corresponding APPROACHING point). It also inherits the properties of *object* and establishes values for some of the variables in the inherited properties.

y:=:{GRASP,MOVE} means that the property (ACTION y v), which is inherited from *object*, has to be instantiated to (ACTION GRASP v) & (ACTION MOVE v).

(x,t):=:(ATTACHED,S1) means that any inherited property containing the tuple (x,t) has to be instantiated by replacing x with ATTACHED and t with S1.

Similarly, *axle* defines the properties common only to axles, but not common to all graspable objects or general objects. It also inherits the properties of its ancestors.

The significance and the advantages of the above description are those generally mentioned in connection with the hierarchies of prototypes: knowledge is organized around relevant conceptual entities (prototypes) in a memory efficient manner (the inheritance mechanism allows for a unique representation of a property common to some objects as the property of a prototype of those objects). Moreover, the generalization techniques are able to reveal subtle features that are common to the objects.

One disadvantage of the above descriptions is that they are somehow complicated. Therefore they need to be simplified by removing certain facts that may be proven to be useless for the application domain.

Once learned, these prototypes may be directly referred to by other pieces of knowledge [Tecuci & al. 1983]. For instance, a rule may indicate to the robot that, for moving a graspable object, it has to move the hand to the object's approaching point, open the hand, move the hand to the object, close the hand, and move the hand. This rule may be used for each graspable object instance.

VII. CONCLUSIONS

One of the most critical problems of inductive learning is that of choosing among competing generalizations. In this appendix we proposed and justified a solution to this problem which is based on the notion of conceptual distance and consists in enhancing the symbolical method of generalization with some numerical estimations.

A distinctive feature of our approach is that learning from examples and learning by observation are seen as complementary learning paradigms:

- learning by observation uses learning from examples to determine the examples to cluster;
- learning from examples uses learning by observation to determine the objects to match.

These relationships allowed the definition of a recursive learning method in which a complex learning from examples task is reduced to a task of clustering the objects contained in the examples, which in turn is reduced to a task of learning from these objects.

BIBLIOGRAPHY

[Anderson, 1986] Anderson J.R.,
Knowledge Compilation: The General Learning Mechanism, in Michalski R.S., Carbonell J.G., Mitchell T.M. (eds), Machine Learning: An Artificial Intelligence Approach, Volume 2, Morgan-Kaufmann, 1986, pp.289-310.

[Anderson, 1986] Anderson J.R.,
Causal Analysis and Inductive Learning, Proceedings of the International Machine Learning Workshop, Irvine, California, 1987, pp.288-299.

[Attardi & Simi, 1986] Attardi G. & Simi M.,
A Description Oriented Logic for Building Knowledge Bases, Technical Report ESP/86/3, Universita di Pisa, April 1986.

[Bareiss & Porter 1987] Bareiss E. & Porter B.,
PROTOS: An Exemplar-Based Learning Apprentice, in Langley P.(ed) Proc. 4th Int. Workshop on Machine Learning, Irvine, 1987.

[Barr & Feigenbaum, 1981] Barr A. & Feigenbaum E. (eds),
The Handbook of Artificial Intelligence, Vol. 1, M. Kaufmann, 1981.

[Benamou & Kodratoff, 1986] N. Benamou N. & Kodratoff Y.,
Conceptual Hierarchical Ascending Classification, Research Report 305, LRI, Université de Paris-Sud, 1986.

[Benjamin & Harrison, 1983] Benjamin P.D., Harrison, M.C.,
A Production System for Learning Plans from an Expert, in Proceedings AAAI-83, Washington, 1983, pp.22-26.

[Bobrow & Winograd, 1977] Bobrow D.G., Winograd T.,
An Overview of KRL, a Knowledge Representation Language, Cognitive Science, 1, 1977.

[Bollinger, 1986] Bollinger T.,
Généralisation et Apprentissage à Partir d'Exemples, Thèse de 3e Cycle, Université de Paris-Sud, 1986.

[Brachman, 1979] Brachman R.J.,
On the Epistemological Status of Semantic Networks, in N.V. Findler (ed), Associative Networks, AP, New York, 1979.

[Brachman, 1983] Brachman R.J.,
What IS-A Is and Isn't: An Analysis of Taxonomic Links in Semantic Networks, Computer, October 1983, pp.30-36.

[Brazdil, 1986] Brazdil P.B.,
Transfer of Knowledge Between Systems: Some Considerations Concerning System's Abilities, in Proceedings of the First European Working Session on Learning, Orsay, February, 1986.

[Bresina, 1981] Bresina J.L.,
An Interactive Planner that Creates a Structured, Annotated Trace of its Operation, Research Report CBM-TR-123, Rutgers University, December 1981.

[Bundy & Silver, 1982] Bundy A., Silver B.,
A Critical Survey of Rule Learning Programs, in Proceedings of ECAI-82, Orsay, pp.151-157.

[Buntine, 1986] Buntine W.L.,
Induction of Horn Clauses: Methods and the Plausible Generalization Algorithm, Technical Report 86.7, New South Wales Institute of Technology and Macquarie University, October, 1986.

[Buntine & Stirling, 1987] Buntine W. & Stirling D.,
Interactive Induction, pp. 13, September 1987, submitted to IEEE applied Artificial Intelligence.

[Burstein 1986] Burstein M. H.
Concept Formation by Analogical Reasoning and Debugging, in Michalski R., Carbonell J. & Mitchell T. (eds) Machine Learning: An Artificial Intelligence Approach, Vol. 2, Morgan Kaufmann 1986, pp.351-370.

[Carbonell, 1983] Carbonell J.G.,
Learning by Analogy: Formulating and Generalizing Plans from Past Experience, in Michalski R.S., Carbonell J.G., Mitchell T.M., (eds), Tioga Publishing Company 1983, pp.137-162.

[Carbonell & al. 1983] Carbonell J.G., Michalski R.S., Mitchell T.M.,
An Overview of Machine Learning,in Michalski R.S., Carbonell J.G., Mitchell T.M., (eds), Tioga Publishing Company 1983, pp.3-24.

[Carbonell, 1986] Carbonell J.,
Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition, in Michalski R., Carbonell J. & Mitchell T. (eds) Machine Learning: An Artificial Intelligence Approach, Vol. 2, Morgan Kaufmann 1986, pp.371-392.

[Carbonell & Gil 1987] Carbonell J. & Gil Y.,
Learning by Experimentation, in Langley P.(ed) Proc. 4th Int. Workshop on Machine Learning, Irvine, 1987.

[Chailloux 1985] Chailloux J.,
LE_LISP de l'INRIA, Le Manuel de Reference, INRIA, Rocquencourt, February, 1985.

[Chisholm & Sleeman, 1979] Chisholm I.H., Sleeman D.H.,
An Aide for Theory Formation, in D. Michie (ed), Expert Systems in the Microelectronic Age, 1979, pp.202-212.

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

[Clavieras, 1984] Clavieras B.,
Modifications de la Représentation des Connaissances en Apprentissage Inductif, Thèse de 3e Cycle, Université de Paris-Sud, 1984.

[Coelho & al. 1980] Coelho H., Cotta J.C., Pereira L.M.,
How to Solve it with PROLOG, Laboratorio National de Engenharia Civil, Lisabona, 1980.

[Cohen & Feigenbaum, 1982] Cohen P. & Feigenbaum E. (eds),
The Handbook of Artificial Intelligence, Vol. 2, 3, M. Kaufmann, 1982.

[Cohen & Sammut, 1984] Cohen B., Sammut C.,
Program Synthesis Through Concept Learning, in Automatic Program Construction Techniques, Biermann A.W., Guiho G., Kodratoff Y. eds, Macmillan Publishing Company, 1984, pp. 463-482.

[Cornuejols, 1988] Cornuejols A.,
INFLUENCE: un Système d'Apprentissage par Adaptation, in Proc. of JFA-88, Cassis, France, 5-6 May, 1988, pp. 39-55.

[Costa, 1986] Costa E.J.,
Artificial Intelligence and Education: The Role of Knowledge in Teaching, in Proceedings of the First European Working Session on Learning, Orsay, February, 1986.

[Daniel & Tate, 1982] Daniel L., Tate A.,
A Retrospective on the "Planning: A Joint AI:OR Approach" Project, DAI Working Paper 125, University of Edinburgh, 1982.

[Davidoviciu, 1983] Davidoviciu A.,
Robotica si inteligenta artificiala, in Inteligenta artificiala si robotica, Ed. Academiei, 1983, pp.214-222.

[Davis, 1979] Davis R.,
Interactive Transfer of Expertise: Acquisition of New Inference Rules, Artificial Intelligence 12 (1979), 121-157.

[Davis & Lenat, 1983] Davis R., Lenat D.B.,
Knowledge-Based Systems in Artificial Intelligence, McGraw-Hill, New York, 1981.

[DeJong, 1981] DeJong G.,
Generalizations Based on Explanations, Proc. 7th IJCAI, 1981, pp. 67-69.

[DeJong & Mooney, 1986] DeJong G., Mooney R.,
Explanation-Based Learning: An Alternative View, Machine Learning 1, pp. 145-176.

[Deliyanni & Kowalski, 1979] Deliyanni A., Kowalski R.A.,
Logic and Semantic Networks, CACM, 22, 3, 1979.

[De Raedt & Bruynooghe, 1988] De Raedt L., Bruynooghe M.,
On Explanation and Bias in Concept Learning, Katholieke Universiteit Leuven, 1988.

[Dietterich, 1980] Dietterich T.G.,
Applying General Induction Methods to the Card Game Eleusis, in Proceedings AAAI-80, pp.218-220.

[Dietterich & Michalski, 1981] Dietterich G.T., Michalski R.S.,
Inductive Learning of Structural Descriptions: Evaluation Criteria and Comparative Review of Selected Methods, Artificial Intelligence Journal 16, 1981, pp. 257-294.

[Doyle, 1986] Doyle R.,
Constructing and Refining Causal Explanations from an Inconsistent Domain Theory, in Proceedings AAAI-86, Philadelphia, 1986, pp.538-544.

[Draganescu, 1980] Draganescu M.,
A doua revolutie industriala. Microelectronica, Automatica, Informatica, Ed. Tehnica, Bucharest,1980.

[Draganescu, 1984] Draganescu M.,
Information, Heuristics, Creation, in Plander I. (ed), Artificial Intelligence and Information-control Systems for Robots, North-Holland, 1984.

[Draghici, 1988] Draghici M.,
A Psycho-Relational Approach to Data Base Conceptual Structuring, PhD Thesis Abstract, Bucharest University, 1988.

[Dufay & Latombe, 1983] Dufay B., Latombe J-C.,
Robot Programming by Inductive Learning, Proceedings IJCAI-83, Karlsruhe, 1983.

[Duval & Kodratoff, 1986] Duval B., Kodratoff Y.,
Automated Deduction in an Uncertain and Inconsistent Data Base, Proceedings ECAI-86, Brighton, 1986, pp.101-108.

[Fahlman, 1974] Fahlman S. E.,
A Planning System for Robot Construction Tasks, Artificial Intelligence, 5, pp. 1-49.

[Farreny, 1980] Farreny H.,
Un Système pour L'Expression et la Résolution de Problemes Orienté vers le Controle de Robots, Thèse d'Etat, Université Paul-Sabatier, 1980.

[Feigenbaum, 1977] Feigenbaum E.,
The Art of Artificial Intelligence: Themes and Case Studies in Knowledge Engineering, Proc. IJCAI 5, 1014-1029, 1977.

[Feigenbaum & Feldman, 1963] Feigenbaum E.A. & Feldman J. (eds),Computers and Thought, New York: McGraw-Hill, 1963.

[Fikes & al. 1972] Fikes R.E., Hart P.E., Nilsson N.J.,
Learning and Executing Generalized Robot Plans, Artificial Intelligence, 3, pp. 251-288, 1972.

[Filmore, 1968] Filmore C.,
The Case for Case, in E. Bach, R. Harms (eds), Universals in Linguistic Theory, New York: Holt, Rinehert and Winston, 1968.

[Fisher & Langley, 1985] Fisher D., Langley P.,
Approaches to Conceptual Clustering, in Proceedings IJCAI-85, Los Angeles, 1985, pp.691-697.

[Forbus & Gentner, 1986] Forbus K., Gentner D.,
Learning Physical Domains: Toward a Theoretical Framework, in Michalski R.S., Carbonell J.G., Mitchell T.M. (eds), Machine Learning: An Artificial Intelligence Approach, Volume 2, Morgan-Kaufmann, 1986, pp.311-348.

[Friedland, 1979] Friedland P.,
Knowledge-Based Experiment Design in Molecular Genetics, Ph. Thesis, Stanford University, august 1979.

[Fu & Buchanan, 1985] Fu L-M., Buchanan B.G.,
Learning Intermediate Concepts in Constructing a Hierarchical Knowledge Base, in Proceedings IJCAI-85, Los Angeles, 1985, pp.659-666.

[Ghallab, 1982] Ghallab M.,
Optimisation de Processus Decisionnels pour la Robotique, Thèse d'Etat, Université Paul Sabatier, Toulouse, 1982.

[Ganascia, 1983] Ganascia J-G.,
Détection des Pannes par Système Expert, Internal Paper, Univ. Paris-Sud, Orsay, 1983.

[Ganascia, 1987] Ganascia J-G.,
AGAPE et CHARADE: deux Techniques d'Apprentissage Symbolique Appliquées à la Construction des Bases de Connaissances. Thèse d'Etat, Université de Paris-Sud, 1987.

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

[Georgescu, 1986] Georgescu I.,
Inteligenta artificiala, Ed. Academiei, 1986.

[Giralt & al. 1979] Giralt G., Sobek R., Chatila R.,
A multilevel Planning and Navigation System for a Mobile Robot: a First Approach to HILARE, Proceedings IJCAI-79, Tokio, 1979.

[Giumale, 1984] Giumale C.A.,
Inference Processes: A Mean to Shape Knowledge Control, in Plander I. (ed), Artificial Intelligence and Information-control Systems for Robots, North-Holland, 1984.

[Giumale & al. 1987] Giumale C.A., Preotescu D., Serbanati L.D., Tufis D., Tecuci G., Cristea D.,
LISP, Ed. Tehnica, Bucharest, 1987.

[Hall, 1986] Hall R., Learning by Failing to Explain, in Proceedings AAAI-86, Philadelphia, 1986, pp.568-573. in Proceedings AAAI-86, Philadelphia, 1986, pp.

[Hayes, 1977] Hayes P.J.,
In Defence of Logic, Proceedings IJCAI-77, Massachussets, 1977.

[Hayes-Roth & al. 1978] Hayes-Roth F., Waterman D., Lenat D.,
Principles of Pattern-Directed Inference Systems, in D.A. Waterman, F. Hayes-Roth (eds), Pattern-Directed Inference Systems, Academic Press, New York, 1978.

[Hayes-Roth, 1985] Hayes-Roth B.,
A Blackboard Architecture for Control, Artificial Intelligence, 26, pp.251-321, 1985.

[Hendrix, 1979] Hendrix G.G.,
Encoding Knowledge in Partitioned Networks, in N.V. Findler (ed), Associative Networks, AP, New York, 1979.

[Holte, 1984] Holte R.,
Artificial Intelligence Approaches to Concept Learning, in I. Alexander (ed), Digital Information Systems, Prentice-Hall International, Englewood Cliffs, N.J., 1984.

[Hutchinson, 1986] Hutchinson A.,
An Inheritance Mechanism, in Proceedings of the First European Working Session on Learning, Orsay, February, 1986.

[Hutchinson, 1986] Hutchinson A.,
A Data Structure and Algorithm for a Self-Augmenting Heuristic Program, The Computer Journal, Vol.29, No.2, 1986, pp.135-150.

[Jappinen, 1981] Jappinen H.,
Sense-Controlled Flexible Robot Behavior, International Journal of Computer and Information Sciences, Vol. 10, No. 2, 1981.

[Kedar-Cabelli, 1985] Kedar-Cabelli S.,
Purpose-Directed Analogy, In Proceedings of the Cognitive Science Society, pp.150-159, Irvine, California, 1985.

[Kodratoff, 1983] Kodratoff Y.,
Generalizing and Particularizing as the Techniques of Learning, Computers and Artificial Intelligence, 4, pp. 417-441, 1983.

[Kodratoff & al. 1984] Kodratoff Y., Ganascia J.G., Clavieras B., Bollinger T., Tecuci G.,
Careful Generalization for Concept Learning, Proc. ECAI-84, Pisa 1984, pp. 483-492. Also in Advances in Artificial Intelligence, T. O'Shea (ed), pp. 229-238, North-Holland Amsterdam 1985.

[Kodratoff, 1985] Kodratoff Y.,
Une théorie et une méthodologie de l'apprentissage symbolique, Actes COGNITIVA 85, Paris, June 1985, pp 639-651.

[Kodratoff, 1986] Kodratoff Y.,
Learning Expert Knowledge and Theorem Proving, in C.R. Rollinger, W. Horn (eds), GWAI-86 und 2 Osterreichische Artificial Intelligence Tagung, Berlin: Springer Verlag, 1986.

[Kodratoff & Ganascia, 1986] Kodratoff Y. & Ganascia J-G.,
Improving the Generalization Step in Learning, in Michalski R., Carbonell J. & Mitchell T. (eds) Machine Learning: An Artificial Intelligence Approach, Vol. 2, Morgan Kaufmann 1986, pp. 215-244.

[Kodratoff & Tecuci, 1986] Kodratoff Y. & Tecuci G.,
Rule Learning in DISCIPOL, Proceedings of the First European Working Session on Learning, Orsay, 1986.

[Kodratoff & Tecuci, 1987a] Kodratoff Y. & Tecuci G.,
Techniques of Design and DISCIPLE Learning Apprentice, International Journal of Expert Systems: Research and Applications, vol.1, no.1, pp. 39-66, 1987.

[Kodratoff & Tecuci, 1987b] Kodratoff Y. & Tecuci G.,
What is an explanation in DISCIPLE, Proceedings of the Forth International Working Session on Learning, Irvine, California, June, 1987, Morgan Kaufmann.

[Kodratoff & Tecuci, 1987c] Kodratoff Y. & Tecuci G.,
DISCIPLE1: Interactive Apprentice System in Weak Theory Fields, Proceedings of the 10th International Joint Conference on Artificial Intelligence, Milan, 1987, Morgan Kaufmann.

[Kodratoff & Tecuci, 1987d] Kodratoff Y. & Tecuci G.,
DISCIPLE: An Integrated Expert and Learning System for Weak Theory Fields, LRI Research Report 371, 40 pg, Orsay, September 1987.

[Kodratoff & Tecuci, 1987e] Kodratoff Y. & Tecuci G.,
The Central Role of Explanations in DISCIPLE. First European Knowledge Acquisition for Knowledge-Based Systems Workshop, Geseke, October 1987.

[Kodratoff & Tecuci, 1988a] Kodratoff Y. & Tecuci G.,
Learning Based on Conceptual Distance, IEEE Transactions on Pattern Analysis and Machine Intelligence, November 1988.

[Kodratoff & Tecuci, 1988b] Kodratoff Y. & Tecuci G.,
Learning with Different Levels of Knowledge, Proc. of the 2nd European Knowledge Acquisition for Knowledge-Based Systems Workshop, Bonn, 19-23 June, 1988.

[Kuipers, 1979] Kuipers B.,
On Representing Commonsense Knowledge, in N.V. Findler (ed), Associative Networks, AP, New York, 1979.

[Laird, 1988] Laird J.E.,
Proc. Fifth Int. Workshop on Machine Learning, Ann Arbor, June, Morgan-Kaufmann, 1988.

[Langley, 1978] Langley P.W.,
BACON.1: A General Discovery System, Proceedings CSCSI, Toronto, 1978, pp.173-180.

[Langley & Carbonell, 1984] Langley P., Carbonell J.G.,
Approaches to Machine Learning, Carnegie-Mellon University, Pennsylvania, 1984.

[Langley & Nordhausen, 1986] Langley P., Nordhausen B.,
A Framework for Empirical Discovery, Proceedings of the International Meeting on Advances in Learning, Les Arcs, 1986.

[Langley, 1987] Langley P. (ed),
Proc. Forth Int. Workshop on Machine Learning, University of California, Morgan-Kaufmann, 1987.

[Lebowitz 1986] Lebowitz M.,
Integrated Learning: Controlling Explanation, Cognitive Science, 10, 1986.

[Lebowitz, 1986] Lebowitz M.,
Concept Learning in a Rich Input Domain: Generalization-Based Memory, in Michalski R.S., Carbonell J.G., Mitchell T.M. (eds), Machine Learning: An Artificial Intelligence Approach, Volume 2, Morgan-Kaufmann, 1986, pp.193-213.

[Levesque & Mylopoulos, 1979] Levesque H., Mylopoulos J.,
A Procedural Semantics for Semantic Networks, in N.V. Findler (ed), Associative Networks, AP, New York, 1979.

[Lieberman & Wesley, 1977] Lieberman L.I., Wesley M.A.,
AUTOPASS: An Automatic Programming System for Computer Controlled Mechanical Assembly, IBM J.Res.Develop., July, 1977.

[Mahadevan, 1985] Mahadevan S.,
Verification-based Learning: A Generalized Strategy for Inferring Problem Reduction Methods, Proc. IJCAI-85, Los Angeles 1985, PP. 616-623.

[Malita & Malita, 1987] Malita M.M., Malita M.,
Bazele Inteligentei Artificiale, Vol.I, Ed. Tehnica, Bucharest, 1987.

[Manago, 1986] Manago M.,
Object Oriented Generalization: A Tool for Improving Knowledge-Based Systems, Proceedings of the International Meeting on Advances in Learning, Les Arcs, 1986.

[Manago & Kodratoff, 1987] Manago M., Kodratoff Y.,
Noise and Knowledge Acquisition, Proceedings of the 10th International Joint Conference on Artificial Intelligence, Milan, 1987, Morgan Kaufmann.

[Mândutianu & al. 1983] Mândutianu D., Tecuci G., Voinea S.,
Programarea Textuala a Robotilor, in Inteligenta Artificiala si Robotica, Ed. Academiei, Bucharest, 1983.

[McCarthy & Hayes, 1969] McCarthy J., Hayes P.J.,
Some Philosophical Problems from the Standpoint of Artificial Intelligence, in Machine Intelligence 6, B. Meltzer, D. Michie (eds), Edinburgh University Press, Edinburgh, 1969.

[Mooney & Bennet, 86] Mooney R. and Bennet S.,
A Domain Independent Explanation-Based Generalizer, Proc. AAAI-86, Philadelphia, pp. 551-555.

[Minsky, 1975] Minsky M.L.,
A Framework for Representing Knowledge, in The Psychology of Computer Vision, Winston P.H. (ed), McGraw-Hill, New-York, 1975, pp. 211-277.

[Michalski, 1980] Michalski R.S.,
Pattern Recognition as Rule-Guided Inductive Inference, Vol. PAMI-2, No.4, pp. 349-361, July 1980.

[Michalski & Chilausky, 1980] Michalski R.S., Chilausky R.L.,
Learning by Being Told and Learning from Examples: An Experimental Comparison of 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, No.4, pp.125-161.

[Michalski, 1983] Michalski R.S.,
A Theory and a Methodology of Inductive Learning, Artificial Intelligence 20(1983), pp 111-161.

[Michalski, Carbonell & Mitchell, 1983] Michalski R., Carbonell J. & Mitchell T. (eds)
Machine Learning: An Artificial Intelligence Approach, Vol. 1, Tioga Publishing Company 1983.

[Michalski & Stepp, 1983a] Michalski R.S., Stepp R.,
Learning From Observation: Conceptual Clustering, in Michalski R.S., Carbonell J.G., Mitchell T.M., (eds), Tioga Publishing Company 1983, pp.331-364.

[Michalski & Stepp, 1983b] Michalski R.S., Stepp R.,
Automated Construction of Classification: Conceptual Clustering Versus Numerical Taxonomy, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.5, No.4, pp.396-410, July 1983.

[Michalski & Stepp, 1983c] Michalski R.S., Stepp R.E.,
How to Structure Structured Objects, Proceedings of the International Machine Learning Workshop, Monticello, Illinois, 1983, pp.156-160.

[Michalski & Baskin, 1983] Michalski R.S., Baskin A.B.,
Integrating Multiple Knowledge Representations and Learning Capabilities in an Expert System: The ADVICE System, in Proceedings IJCAI-83, Karlsruhe, 1983, pp.256-258.

[Michalski, 1984] Michalski R.S.,
Inductive Learning as Rule-guided Transformation of Symbolic Descriptions: a Theory and Implementation, in *Automatic Program Construction Techniques*, A. W. Biermann, G. Guiho and Y. Kodratoff (eds), Mac-Millan Publishing Company, 1984, pp. 517-552.

[Michalski, 1986] Michalski R.S.,
Understanding the Nature of Learning: Issues and Research Directions, in Michalski R., Carbonell J. & Mitchell T. (eds) Machine Learning: An Artificial Intelligence Approach, Vol. 2, Morgan Kaufmann 1986, pp. 3-25.

[Michalski, 1986] Michalski R.S.,
Inference-based Theory of Learning, Proc. International Meeting on Advances in Learning, Les Arcs, 1986.

[Michalski & Winston, 1986] Michalski R.S., Winston P.H.,
Variable Precision Logic, Artificial Intelligence 29 (1986), pp.121-146.

[Michalski, Carbonell & Mitchell, 1986] Michalski R., Carbonell J. & Mitchell T. (eds)
Machine Learning: An Artificial Intelligence Approach, Vol. 2, Morgan Kaufmann 1986.

[Michalski & al. 1986] Michalski R.S., Mozetic I, Hong J, Lavrac N.,
The AQ15 Inductive Learning System: An Overview and Experiments, Internal Paper, University of Illinois at Urbana-Champaign, 1986.

[Michie, 1974] Michie D.,
On Machine Intelligence, Edinburgh University Press, 1974.

[Michie, 1979] Michie D. (ed),
Expert Systems in the Microelectronic Age, 1979.

[Michie, 1982] Michie D.,
'Man-like' Capabilities in Computers: A note on Computer Induction, Cognition, 12 (1982), pp.97-108.

[Minsky, 1975] Minsky M.,
A Framework for Representing Knowledge, in P.H. Winston (ed), The Psychology of Computer Vision, New-York: McGraw-Hill, 1975.

[Minton, 1985] Minton S.,
Selectively Generalizing Plans for Problem Solving, in Proceedings IJCAI-85, Los Angeles, 1985, pp.596-599.

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

[Mitchell & al. 1981] Mitchell T.M., Carbonell J.G., Michalski R.S.,
Special Section on Machine Learning, SIGART Newsletter, No.76, pp.25-64, April 1981.

[Mitchell, 1982] Mitchell T.M.,
Generalization as Search, Artificial Intelligence, Vol.18, No.2, pp.203-226, March 1982.

[Mitchell, 1983] Mitchell T.M.,
Learning and Problem Solving, Proceedings IJCAI-83, Karlsruhe, 1983, pp.1139-1151.

[Mitchell & al. 1983] Mitchell T.M., Utgoff P.E., Banerji R.,
Learning by Experimentation, Acquiring and Refining Problem-Solving Heuristics, in Machine Learning: An Artificial Intelligence Approach, Michalski R.S., Carbonell J.G., Mitchell T.M., (eds), Tioga Publishing Company 1983, pp. 163-190, now distributed in the USA by Morgan Kaufmann and in Europe by Springer Verlag.

[Mitchell, Carbonell & Michalski, 1985] Mitchell T., Carbonell J. and Michalski R. (eds),
Machine Learning: A Guide to Current Research, Kluwer Academic Publishers, 1985.

[Mitchell & al. 1985] Mitchell T., Mahadevan S. and Steinberg L.,
LEAP: a Learning Apprentice System for VLSI Design, Proc. IJCAI-85, Los Angeles 1985, 573-580.

[Mitchell & al. 1986] Mitchell T.M., Keller R.M., Kedar-Cabelli S.T., Explanation-Based Generalization: A Unifying View, Machine Learning 1, pp. 47-80, 1986.

[Mooney & DeJong, 1985] Mooney R., DeJong J.,
Learning Schemata for Natural Language Processing, in Proceedings IJCAI-85, Los Angeles, 1985, pp.681-687.

[Mooney & Bennet, 1986] Mooney R., Bennet S.,
A Domain Independent Explanation Based Generalizer, in Proceedings AAAI-86, Philadelphia, 1986, pp.551-555.

[Nilsson 1971] Nilsson N.,
Problem Solving Methods in Artificial Intelligence, McGraw-Hill, 1971.

[Nilsson, 1978] Nilsson N.J.,
Some Examples of AI Mechanisms for Goal Seeking, Planning, and Reasoning, in F. Klix (ed), Human and Artificial Intelligence, Berlin, 1978.

[Nilsson, 1980] Nilsson N.J.,
Principles of Artificial Intelligence, Tioga, Palo Alto, California, 1980.

[Pazzani & al. 1986] Pazzani M., Dyer M., Flowers M.,
The Role of Prior Causal Theories in Generalization,in Proceedings AAAI-86, Philadelphia, 1986, pp.545-550.

[Petrescu, 1983] Petrescu M.,
Relatii Vagi si Baze de Date, The 5-th International Conference on Control Systems and Computer Science, Bucharest, 1983, pp.149-155.

[Petrescu & Tecuci, 1986] Petrescu M. & Tecuci G.,
Sisteme expert instruibile in proiectarea asistata de calculator, A 4-a sesiune nationala de teoria sistemelor, Craiova 1986.

[Petrescu & Tecuci, 1987] Petrescu M. & Tecuci G.,
Integrating Learning with Expert Systems, The 7-th International Conference on Control Systems and Computer Science, Bucharest, May 1987.

[Quillian, 1968] Quillian M.R.,
Semantic Memory, in Semantic Information Processing, Minsky M., editor, Cambridge, Mass: MIT Press, 1968, pp. 227-270.

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

[Rajamoney, 1986] Rajamoney S.A.,
Automated Design of Experiments for Refining Theories, Master of Science Thesis, University of Illinois at Urbana-Champaign, 1986.

[Robinson & Wilkins, 1981] Robinson A., Wilkins D.,
An Interactive Planning System, Technical Note 245, SRI International, July, 1981, 23 pg.

[Russel, 1987] Russel S.J.,
Analogy and Single-Instance Generalization, in Langley P.(ed) Proc. 4th Int. Workshop on Machine Learning, Irvine, 1987.

[Rychener, 1984] Rychener M.D,
The Instructible Production System: a Retrospective Analysis, in Machine Learning: An Artificial Intelligence Approach, Michalski R.S., Carbonell J.G., Mitchell T.M., (eds), Tioga Publishing Company 1983, pp. 163-190.

[Sacerdoti 1975] Sacerdoti E.,
A Structure for Plans and Behavior, Technical Note 109, SRI International, August 1975.

[Sammut & Banerji, 1986] Sammut C., Banerji R.B.,
Learning concepts by asking questions, in Machine Learning: An Artificial Intelligence Approach, Volume 2, Michalski R.S., Carbonell J.G., Mitchell T.M. (eds), Morgan-Kaufmann 1986, pp. 167-191.

[Sammut & Hume, 1987] Sammut C., Hume D.,
Observation and Generalization in a Simulated Robot World, in Langley P.(ed) Proc. 4th Int. Workshop on Machine Learning, Irvine, 1987.

[Samuel, 1959] Samuel A. L.,
Some Studies in Machine Learning Using the Game of Checkers, IBM Journal of Research and Development, No. 3, pp. 210-220, 1959.

[Silver, 1986] Silver B.,
Precondition Analysis: Learning Control Information, in Machine Learning: An Artificial Intelligence Approach, Volume 2, Michalski R.S., Carbonell J.G., Mitchell T.M. (eds), Morgan-Kaufmann 1986.

[Simon & Lee, 1974] Simon H., A. and Lee G.,
Problem Solving and Rule Induction: A Unified View, in Gregg L. (ed), Knowledge and Cognition, Hillsdale, N.J.: Lawrence Erlbaum, 105-127.

[Simon, 1983] Simon H.,
Why Should Machines Learn ?, in Michalski R.S., Carbonell J.G., Mitchell T.M., (eds), Tioga Publishing Company 1983, pp.25-38.

[Siqueira & Puget, 1988] Siqueira J.L.N., Puget J.-F.,
Explanation-Based Generalization of Failures, To appear in Proc. ECAI, 1988.

[Sleeman, 1987] Sleeman D.,
Some Challenges for Intelligent Tutoring Systems, Proceedings IJACI-87, Milan, 1987, pp.1166-1168.

[Sleeman & Hirsh, 1987] Sleeman D., Hirsh H.,
Research Report, Edinburgh University, 1987.

[Smith & al. 1977] Smith R.G., Mitchell T.M., Chestek R.A., Buchanan B.G.,
A Model for Learning Systems, Proceedings IJCAI-77, Massachussets, 1977, pp.338-343.

[Smith & al. 85] Smith R., Winston H., Mitchell T. and Buchanan B.,
Representation and Use of Explicit Justifications for Knowledge Base Refinement, Proc. IJCAI-85, Los Angeles.

[Soloway & Riseman, 1977] Soloway E.M., Riseman E.M.,
Levels of Pattern Description in Learning, Proceedings IJCAI-77, Massachussets, 1977, pp.801-811.

[Sridharan & Bresina, 1982] Sridharan N. and Bresina J.,
Plan Formation in Large Realistic Domains, Proc. CSCSI Conference, Saskatoon, Saskatchewan 1982, 12-18.

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

[Stefik, 1980] Stefik M.,
Planning with Constraints (MOLGEN: Part 1), in Artificial Intelligence 14, no.2, September 1980, pp.111-139.

[Stefik, 1980] Stefik M.,
Planning and Meta-Planning (MOLGEN: Part 2), in Artificial Intelligence 14, no.2, September 1980, pp.141-170.

[Stefik & al. 1982] Stefik M., Aikins J., Balzer R., Benoit J., Birnbaum L., Hayes-Roth F., Sacerdoti E.,
The Organization of Expert Systems, A Tutorial, Artificial Intelligence, 18 (1982), pp.135-173.

[Steels & Van de Welde, 1985] Steels L., Van de Welde W,
Learning in Second Generation Expert Systems, in J.S. Kowalik (ed), Knowledge-based Problem Solving, New Jersey: Prentice-Hall, 1985.

[Tangwongsan & Fu, 1979] Tangwongsan S., Fu K.S.,
An Application of Learning to Robotic Planning, International Journal of Computer and Information Sciences, Vol. 8, No. 4, 1979.

[Tate, 77] Tate A.,
Generating Project Networks, Proc. IJCAI-77, Massachusetts, pp. 888-893.

[Tecuci, 1981] Tecuci G.,
H-Graphs and their Applications to Pattern Recognition, Buletinul I.P.Bucuresti, Seria Electrotehnica, no.3, 1981, pp.23-34.

[Tecuci & al. 1983] Tecuci G., Mândutianu D., Voinea S.,
A Hierarchical System for Robot Programming, Computers and Artificial Intelligence, 2, 1983.

[Tecuci, 1983] Tecuci G.,
Sistem de Planificare a Robotilor Industriali pentru Operatii Complexe, Buletinul Roman de Informatica, no.1, 1983.

[Tecuci, 1984a] Tecuci G.,
Learning Hierarchical Descriptions from Examples, Computers and Artificial Intelligence 3, 211-222, 1984.

[Tecuci, 1984b] Tecuci G.,
A Framework for Constructing Knowledge-based Planning Systems, in Plander I. (ed), Artificial Intelligence and Information-control Systems for Robots, North-Holland, 1984.

[Tecuci, 1985] Tecuci G.,
SIPLAN: Un Sistem de Planificare a Actiunilor cu Posibilitati de Invatare, Buletinul Roman de Informatica, 1, 1985.

[Tecuci & al. 1986] Tecuci G., Gutu S., Burducea N., Bodnaru Z.,
Proiectarea Tehnologiilor cu Sistemul Siplan, Buletinul Roman de Informatica si Tehnica de Calcul, 1, 1986.

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

[Tecuci, 1987] Tecuci G.,
Steps Towards Second Generation Expert Systems, Proceedings of the Forth International Conference on Artificial Intelligence and Information-control Systems for Robots, Smolenice, October 1987.

[Tecuci, 1988] Tecuci G.,
Mediu de dezvoltare a sistemelor expert instruibile pentru proiectarea asistata de calculator, PhD Thesis, Polytechnical Institute of Bucharest, 1988.

[Tenenberg, 1986] Tenenberg J.,
Planning with Abstraction, in Proceedings AAAI-86, Philadelphia, 1986, pp.76-80.

[Tufis & Cristea, 1985] Tufis D., Cristea D.,
IURES - A Human Engineering Approach to Natural Language Question Answering Systems, in W. Bibel, B. Petkoff (eds): Artificial Intelligence Methodology, Systems, Applications, North-Holland, 1985.

[Tufis, 1986] Tufis D.,
IURES-V2: A System to Aid in Building Natural Language Interfaces, in Proceedings on the International Conference on Advanced Dialog Systems and Natural Language Understanding, Suwalki, Poland, 1986.

[Utgoff, 1986] Utgoff P. E.,
Shift of Bias for Inductive Concept Learning, in Machine Learning: An Artificial Intelligence Approach, Volume 2, Michalski R.S., Carbonell J.G., Mitchell T.M. (eds), Morgan-Kaufmann 1986, pp. 107-148.

[Van Ryzin, 1977] Van Ryzin J.(ed),
Classification and Clustering, Academic Press, New York, 1977.

[Van Someren, 1986] Van Someren M.W.,
Constructive Induction Rules: Reducing the Description Space for Rule Learning, in Proceedings of the First European Working Session on Learning, Orsay, February, 1986.

[Vere, 1978] Vere S.A.,
Inductive Learning of Relational Productions, in D.A. Waterman, F. Hayes-Roth (eds), Pattern-Directed Inference Systems, Academic Press, New York, 1978.

[Vrain, 1987] Vrain, C.,
Un Outil de Généralisation Utilisant Systématiquement les Théorèmes: Le Système OGUST, Thèse, Université de Paris-Sud, 1987.

[Waldinger, 1977] Waldinger R.,
Achieving Several Goals Simultaneously, in E.W. Elcock, D. Michie (eds), Machine Intelligence 8, New York: Halstead/Wiley.

[Waterman & Hayes-Roth, 1978] Waterman D, Hayes-Roth F., (eds) Pattern-Directed Inference Systems, Academic Press, New York, 1978.

[Weber & Nilsson, 1981] Weber B.L., Nilsson N.J. (eds),
Readings in Artificial Intelligence,Tioga, Palo Alto, California, 1981.

[Wilenski, 1980] Wilenski R.,
Meta-Planning, Proceedings of AAAI-80, Stanford, 1980, pp.334-336.

[Wilkins, 1984] Wilkins D.,
Domain Independent Planning: Representation and Plan Generation, Artificial Intelligence 22, pp 269-301.

[Winston, 1970] Winston P.H.,
Learning Structural Descriptions from Examples, Rep. TR-231, MIT. Also available in P.H. Winston (ed), The Psychology of Computer Vision, New-York: McGraw-Hill, 1975.

[Winston, 1977] Winston P.H.,
Artificial Intelligence, New York: Addison Wesley, 1977.

[Winston, 1980] Winston P.H.,
Learning and Reasoning by Analogy, CACM 23, no. 12, pp. 689-703.

[Winston & Horn, 1981] Winston P.H., Horn B.,
LISP, Addison-Wesley, Massachussets, 1981.

[Winston & al. 1983] Winston P.H., Katz B., Binford T., Lowry M.,
Learning Physical Descriptions from Functional Definitions, Examples and Precedents, in Proceedings AAAI-83, Washington, 1983, pp.433-439.

[Winston, 1986] Winston P.H.,
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 2, Morgan-Kaufmann 1986, pp. 45-61.

Nom:
Gheorghe TECUCI

Titre:
DISCIPLE: une Théorie, une Méthodologie et un Système pour Apprendre des Connaissances Expertes.

Résumé:
DISCIPLE est un système qui illustre une théorie et une méthodologie d'apprentissage des connaissances expertes. Il est composé d'un système expert et d'un système d'apprentissage qui utilisent une même base de connaissances. DISCIPLE part de connaissances élémentaires sur un domaine d'application (une théorie du domaine) et, au cours de sessions interactives de résolution de problèmes, apprend des règles générales à partir des solutions spécifiques fournies par l'expert humain. La méthode de résolution de problèmes combine la réduction de problèmes, l'utilisation de contraintes et l'analogie. Quant à l'apprentissage, DISCIPLE utilise différentes méthodes, en fonction de ses connaissances sur la solution de l'utilisateur. Cette solution est considérée comme un exemple pour apprendre une règle générale. Dans le cas d'une théorie complète sur l'exemple, DISCIPLE apprend à partir d'explications, ce qui augmente son efficacité. Dans le cas d'une théorie faible, il intègre l'apprentissage à partir d'explications, l'apprentissage par analogie et l'apprentissage empirique, développant ainsi sa compétence. Enfin, dans le cas d'une théorie incomplète, il apprend en combinant les deux méthodes précédentes, ce qui améliore tant sa compétence que son efficacité.

Mots clés:
Apprentissage, Systèmes Experts, Intelligence Artificielle, Acquisition de Connaissances, Apprentissage à Partir d'Explications, Apprentissage par Analogie, Apprentissage Empirique, Regroupement Logique