In Y.Kodratoff (ed), Machine Learning - EWSL91, Springer-Verlag, 1991

A MULTISTRATEGY LEARNING APPROACH TO DOMAIN MODELING AND KNOWLEDGE ACQUISITION

Gheorghe Tecuci
Center for Artificial Intelligence, Department of Computer Science, George Mason University,
4400 University Drive, Fairfax, VA 22030, USA. Email: tecuci@gmuvax2.gmu.edu

Abstract

This paper presents an approach to domain modeling and knowledge acquisition that consists of a gradual and goal-driven improvement of an incomplete domain model provided by a human expert. Our approach is based on a multistrategy learning method that allows a system with incomplete knowledge to learn general inference or problem solving rules from specific facts or problem solving episodes received from the human expert. The system will learn the general knowledge pieces by considering all their possible instances in the current domain model, trying to learn complete and consistent descriptions. Because of the incompleteness of the domain model the learned rules will have exceptions that are eliminated by refining the definitions of the existing concepts or by defining new concepts.

Keywords: domain modeling, knowledge acquisition, multistrategy learning, rule and concept learning

1. Motivation and related work

The behavior of an expert system is based on an internal model of a real world domain. The model is composed of representations of different entities from the real world, as well as of procedures for manipulating these representations. In order to solve a real world problem, the user has to represent it into the language of the domain model. Then the system will look for a solution by using the model and will show the found solution to the user which will interpret it in the real world. The better this model approximates the real world domain, the more adequate is the system's behavior. Traditionally, such a domain model is built by the knowledge engineer and the human expert. With few exceptions (Morik, 1989), the current knowledge acquisition tools support the modeling task only by helping the human expert to express his knowledge (Boose, Gains and Ganascia, 1989). The built domain model is often only a crude approximation of the represented domain: it incorporates defaults, omits details, and abstracts the represented entities. The causes for this situation are multiple: the representation language is inherently imprecise, the information from the human expert is incomplete, the represented domain has not a well defined theory (Bhatnagar and Kanal, 1986). To cope with these problems, the human expert is often asked to express his knowledge in the form of uncertain knowledge pieces, for instance,

in the form of uncertain rules characterized by certainty factors which are more or less justified. This, however, results in a degradation of the knowledge provided by the human expert because he is asked questions to which he does not know precise answers. Moreover, because the resulting expert system lacks the capability of self-improving its knowledge through experience, the domain model has to be, from the very beginning, complete enough and correct enough for determining reasonable functioning of the system. All of these make the current approaches to building expert systems complex, time-consuming and error-prone.

We believe that the methods and the techniques developed in the field of machine learning (e.g., Michalski, Carbonell and Mitchell, 1983, 1986; Segre, 1989; Kodratoff and Michalski, 1990; Porter and Mooney, 1990) are applicable for partially automating the domain modeling and knowledge acquisition process (Morik, 1989; Wrobel, 1989). For instance, by using empirical induction, a system can learn general concepts or rules characterizing a set of examples. By applying analogical learning, a system may acquire knowledge about an unknown entity by transferring and modifying prior knowledge about a similar entity. By using explanation-based learning, a system may transform inefficient knowledge into efficient rules or concepts. Until now, however, these single-strategy learning methods did not have a significant impact on the field of knowledge acquisition because each strategy requires specific conditions in order to be applicable. For instance, empirical learning typically needs many input examples, though it does not need much background knowledge. Explanation-based learning needs only one example, but requires complete background knowledge. Learning by analogy needs background knowledge analogous with the input. Real-world applications rarely satisfy the requirements of single-strategy learning methods. This explains an increasing interest in building systems that integrate different learning strategies (e.g., Lebowitz, 1986; Wilkins et al., 1986; Tecuci et al., 1987; Minton et al., 1987; Danyluk, 1987; Pazzani, 1988; Dietterich and Flann, 1988).

In this paper we propose an approach to domain modeling and knowledge acquisition based on a synergistic integration of different learning strategies: explanation-based learning, learning by analogy, empirical inductive learning, learning by asking questions and by being told, abduction and conceptual clustering.

2. Toward a methodology for domain modeling and knowledge acquisition

One may distinguish two phases in the development of a domain model. The first one consists of defining a suitable framework for modeling the domain, by choosing a knowledge representation formalism and an associated problem solving method. The second one consists of effectively building the model by representing the entities of the application domain, in the defined framework.

An expert system shell, like EMYCIN (van Melle et al., 1981) for instance, is a framework for representing diagnostic models. Research in expert systems has elaborated different frameworks for different expertise domains like planning, design, diagnosis, monitoring, prediction, interpretation, and expert system shells for such expertise domains have been built.

Choosing (or building) a suitable expert system shell solves the first part of building the model of an application domain. Effectively building the model with the expert system shell represents the second and the much more difficult part of modeling.

The methodology we are presenting is concerned with the automation of the second part of modeling. This methodology is a development of our previous work on learning expert knowledge (Tecuci, 1988; Tecuci and Kodratoff, 1990) and is experimentally implemented in a learning system that may be associated with an expert system shell. The goal of the learning system is to learn directly from the human expert. In other words, the traditional role of the knowledge engineer is taken by the system itself that is building and improving the domain model through successive interactions with the expert. A direct consequence of this goal is that the interaction with the human expert should be as natural for the human expert as possible. A human expert may provide an elementary description of his domain. He is particularly good at providing solutions to problems and to judge if a solution to a problem is good or not. He is less good at providing explanations of why the solutions are good or not but can easily accept or reject tentative explanations. What is particularly difficult for the human expert is to provide general pieces of information as, for instance, general problem solving rules. It is, therefore, the task of the learner to learn such general pieces of information and to iteratively develop and update the world model. Such an update is done through an interaction with the expert in which the expert is asked only the types of questions he is expected to answer correctly.

The scenario for building the domain model is the following one. First, the human expert will define (within the chosen framework) an initial model of his domain. Because he is requested to define only that knowledge which he may easily express, we assume that this initial world model is incomplete. In general, it will consist of incomplete descriptions of some basic object concepts from the domain to be modeled, object concepts that define an initial language for representing new object concepts, facts, rules etc. This initial model will allow the system to learn new knowledge from the expert and thus to develop its model. The validity of the learned knowledge strongly depends of the validity of the initial knowledge. Therefore it is preferable to start with few and valid knowledge than to start with much and imperfect knowledge.

Once an initial model has been defined, the system may react to new inputs from the expert (or, in general, from the real world) with the goal of developing and updating the model so that to become consistent with the inputs. Whenever the system receives an input, it will try to understand and assimilate it into the world model. For instance, if the input is a new fact, the system will try to justify that it is a consequence of the knowledge explicitly represented into the model. To this purpose, it may need to update the model (by abducting new facts or rules, or by explicitly storing the input). Based on the understanding of the input fact, the system may learn a general inference rule allowing the direct derivation of the input (as well as of other related facts). If the input is a problem solving episode, then the system will try to explain to itself the validity of this episode (by building a proof or at least a justification of it) and based on this understanding it may learn a general problem solving rule. If the input consists of several examples of a concept, the system will try to understand the commonalties of these examples in the context of its background knowledge, thus learning the definition of the concept. In order to improve the consistency of the domain model, the system will learn the general knowledge pieces by considering all their possible instances in the current domain model, trying to learn complete

and consistent descriptions. However, because of the incompleteness of the domain model (that, for instance, does not contain some necessary concepts), the learned rules will have exceptions. For instance, they may cover invalid problem solving episodes. Therefore, in order to eliminate the exceptions of the learned rules, new concepts have to be defined, or the definitions of the existing concepts have to be refined. In this way, the domain model is iteratively developed in a goal-driven manner.

The next sections illustrate this methodology with a very simple example from robotics.

3. Modeling the world of a simple robot

We shall briefly illustrate the model building methodology with a very simple example of a robot able to perform simple domestic tasks. The robot receives commands from its master and executes them. A command is executed by performing a single robot action or a sequence of such actions.

3.1 The framework for the domain model

One may model the world of such a robot in terms of object-concepts, states, action-concepts, and problem solving rules.

An object-concept represents a set of objects having similar properties and relations to other object-concepts. The object-concepts may be of different degrees of generality.

An instance of an object-concept represents a specific object from the real world, together with its properties and relations to other objects. One may assimilate an instance with a very specific concept representing only one element.

A specification of all the objects in the world, together with their current properties and relations, represents the current state of the world.

The action-concepts are representations of the actions by which the robot can change the state of the real world. Such an action-concept is represented by the following elements:

- the name of the action and the object-concepts that may have certain roles in the action (the object on which the action is performed, the instrument used, etc.);
- the preconditions: the set of states in which the action may be applied;
- the effects: the state resulted after the execution of the action.

We distinguish between elementary actions and complex actions. An elementary action is directly executable by the robot. A complex action is one that is executed by a sequence of elementary actions.

A problem solving rule is a kind of schemata that indicates a decomposition of a complex action into simpler actions.

The robot receives commands for executing actions. If such an action is an elementary one then the robot is executing it. If the action is a complex one, then the robot has to first decompose it into a sequence of elementary actions.

All these elements constitute the framework for representing the model of the robot world. To build the model, one has to effectively define the object-concepts, the action-concepts, and the problem solving rules.

In the following sections we shall show how such a model is incrementally developed by the human trainer and the learning system.

3.2 Providing an initial domain model

First, the human expert defines the initial model of the domain. This model may be, for instance, the one represented in figure 1.

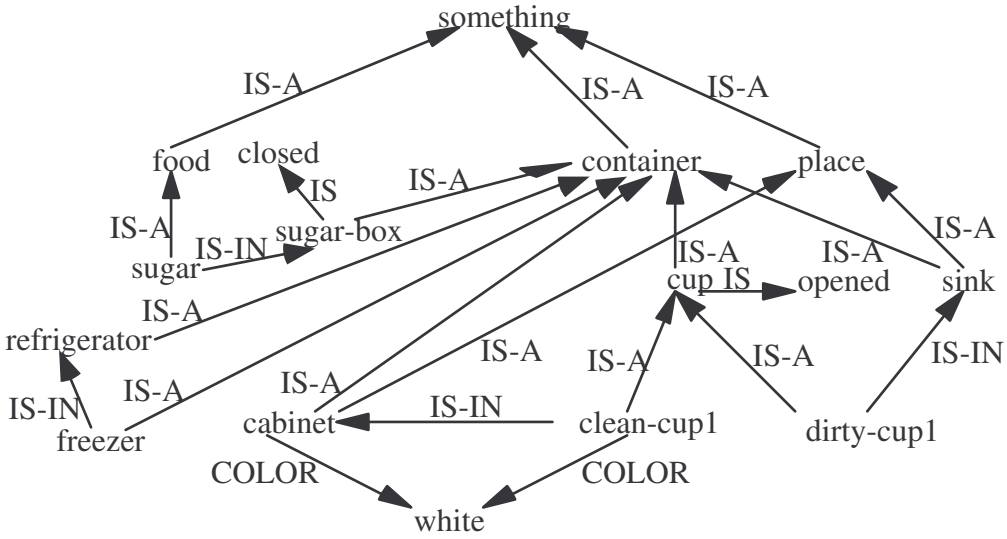

Figure 1. The initial domain model.

This initial model is composed of incomplete descriptions of concepts representing some of the objects from the robot world.

3.3 Incremental development of the model

Once the expert has provided the system with an initial model, he will start teaching it to solve problems by presenting examples of problem solving episodes like the one in figure 2. From such a problem solving episode the system will try to learn a general problem solving rule of the form presented in figure 3.

Example1:

```
The problem                              ; to take clean-cup1
        TAKE clean-cup1                  ; the robot has to
has the solution                         ; open the cabinet
        OPEN cabinet                ; and to take the cup
        TAKE clean-cup1 FROM cabinet     ; from it
```

Figure 2. An example of problem solving episode.

```
IF                                       ; If x and y are two
        x and y satisfy certain constraints     ; objects satisfying
THEN                                ; certain constraints
the problem                              ; then to TAKE x
        TAKE x                           ; the robot has to
has the solution                         ; OPEN y and to
        OPEN y                           ; TAKE x FROM y
        TAKE x FROM y
```

Figure 3. The general rule to be learned from **Example 1**.

As a by-product of learning such a rule the domain model may be improved by:
- completing the definitions of the object-concepts that cover positive or negative instances of the rule's variables (as shown in section 3.5.1);
- defining new object-concepts that cover positive instances of the rule's variables (as shown in section 3.5.2);
- improving the models of the actions from the rule (as shown in section 3.6).

Therefore, rule learning is in fact an opportunity for a goal-driven improvement of the domain model.

3.4 Learning problem solving rules

Table 1 presents the method of learning a general problem solving rule starting from one example. A detailed description of this learning method is given in (Tecuci and Kodratoff, 1990). Therefore here we shall only briefly illustrate it.

As mentioned in Table 1, the learning steps depend of the system's current knowledge. In this section we shall suppose that all system's knowledge is the one from figure 1. In section 3.6, however, we shall show how learning proceeds when the domain model contains also (incomplete) descriptions of the actions "TAKE x", "OPEN y", and "TAKE x FROM y".

First the system will try to understand the problem solving episode received from the expert (Example1), where by understanding we mean proving (or at least justifying) its correctness. It uses heuristics to propose plausible pieces of explanations in terms of the features and the relationships

between the objects from the problem solving episode (Tecuci and Kodratoff, 1990), pieces of explanations requiring the user's validation, as shown in figure 4.

Table 1. The rule learning method.

• *Find an explanation of the validity of the input*
Depending of the system's knowledge, the process of finding the explanation may involve deduction, induction and/or analogy.

• *Generalize the found explanation to an analogy criterion*
Depending of the system's knowledge, the analogy criterion is an inductive, a deductive or an inductive and deductive generalization of the explanation.

• *Use the analogy criterion to generate examples analogous with the input*
The instances of the analogy criterion in the domain model represent explanations similar with the explanation of the input. From each such explanation the system may generate a problem solving episode analogous with the input. These episodes represent positive examples (if they are correct) or negative examples (if they are not correct) of the rule to be learned.

• *Learn from the generated examples*
Learn a general rule that covers as many of the positive examples as possible and as few of the negative examples as possible.

The problem solving episode is correct because:
(clean-cup1 IS-IN cabinet) ? Yes
(clean-cup1 COLOR white) & (cabinet COLOR white) ? No

Figure 4. Finding a justification of Example1.

The found piece of explanation is (clean-cup1 IS-IN cabinet). This explanation is inductively generalized by turning all the contained objects into variables and this generalization is taken as an analogy criterion. The instances of this analogy criterion in the domain model are similar explanations that may account for problem solving episodes similar with Example1. Because analogy is a weak inference, these problem solving episodes could be, however, correct or incorrect (see figure 5). The goal of the system is to learn a general rule that covers the correct problem solving episodes and rejects the incorrect ones.

First of all, from the initial problem solving episode, its explanation and the analogy criterion, the system builds an initial version space for the rule to be learned. This version space is shown in figure 6. As may be noticed, this representation keeps all the knowledge that may be useful for learning the rule: the initial example, the explanation, and the features of the covered objects that are not relevant for the rule learning (and should not be used in the condition of the rule).

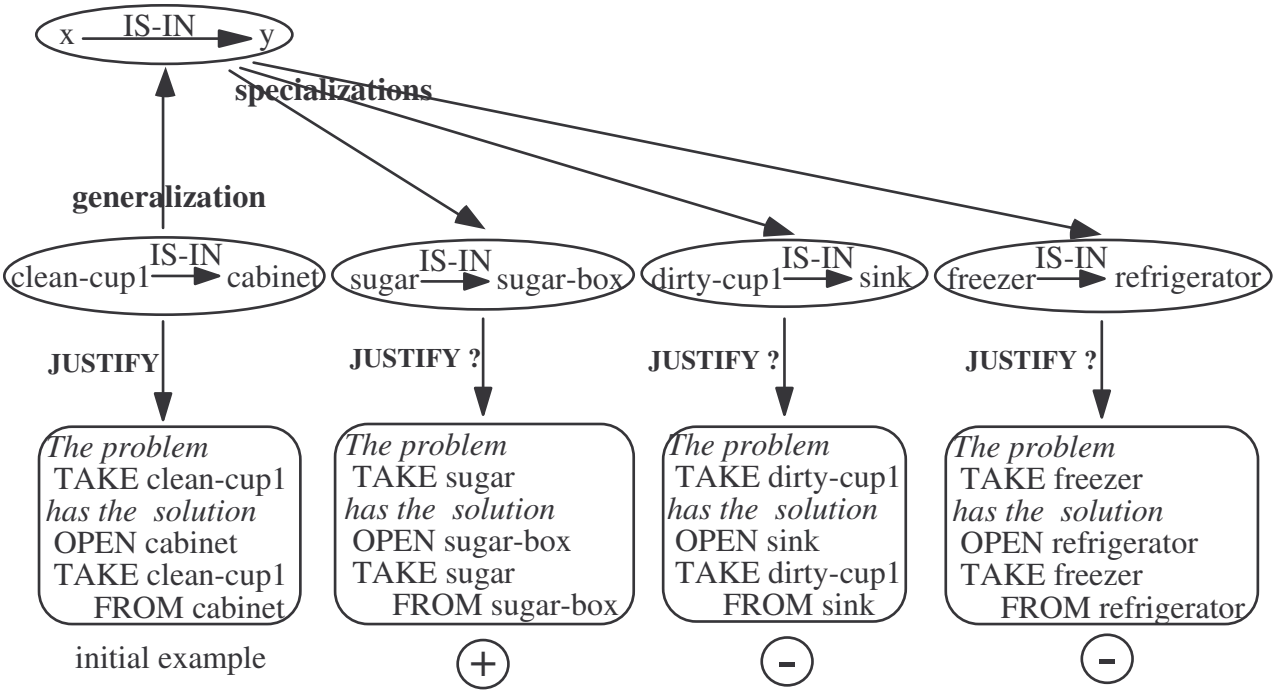

Figure 5. Generation of problem solving episodes analogous with the input one.

IF
upper bound
(x IS-A something) & (y IS-A something) & (x IS-IN y) ; the analogy criterion
lower bound
(x IS-A clean-cup1) & (y IS-A cabinet) & (x IS-IN y) ; the explanation
THEN
the problem
TAKE x
has the solution
OPEN y
TAKE x FROM y
with the positive example:
(x IS-A clean-cup1) & (y IS-A cabinet)
with the irrelevant features:
(x COLOR z) & (y COLOR z)

Figure 6. The initial version space for the rule to be learned.

Next the system applies the analogy criterion to the domain model in figure 1 and generates, one after the other, the problem solving episodes from figure 5, asking the expert to validate them:

Let us consider the problem
TAKE sugar
Is the following a correct solution
OPEN sugar-box
TAKE sugar FROM sugar-box ? Yes

Figure 7. A problem solving episode generated by the system.

Each such generated problem solving episode is used to shrink the version space from figure 6 (Tecuci and Kodratoff, 1990).

For each positive example there are two possible cases:

- if the current upper bound is more general than the positive example, then generalize the lower bound of the rule, as little as possible, so that to cover the example and to remain less general than the upper bound;
- if the current upper bound does not cover the positive example then keep this example as a positive exception of the rule being learned.

For each negative example the system tries to find an explanation of the failure. If such an explanation is found then it is used to particularize both bounds of the current version space for no longer covering the negative example. If no explanation is found then two cases are possible:

- if the current lower bound does not cover the negative example then particularize the upper bound as little as possible so that not to cover the negative example and to remain more general than the lower bound;
- if the current lower bound covers the negative example then keep it as a negative exception of the rule being learned.

Let us suppose that the problem solving episodes from figure 5 are generated in the order from the left to the right. The first generated problem solving episode is accepted by the user and is therefore a new positive example for the rule to be learned. Its explanation (sugar IS-IN sugar-box) is expressed in terms of the variables 'x' and 'y' as

(x IS-A sugar) & (y IS-A sugar-box) & (x IS-IN y)

and is used to generalize the lower bound in figure 6 to

new lower bound
(x IS-A something) & (y IS-A container) & (x IS-IN y)

The next two problem solving episodes generated by the system are rejected by the user. However, their corresponding explanations

(x IS-A dirty-cup1) & (y IS-A sink) & (x IS-IN y)

(x IS-A freezer) & (y IS-A refrigerator) & (x IS-IN y)

are covered by the *new lower bound* of the version space. Therefore these two instances represent negative exceptions for the rule being learned:

IF
upper bound
(x IS-A something) & (y IS-A something) & (x IS-IN y)
lower bound
(x IS-A something) & (y IS-A container) & (x IS-IN y)
THEN
the problem
TAKE x
has the solution
OPEN y
TAKE x FROM y
with the positive examples:
(x IS-A clean-cup1) & (y IS-A cabinet)
(x IS-A sugar) & (y IS-A sugar-box)

with the negative exceptions:
(x IS-A dirty-cup1) & (y IS-A sink)
(x IS-A freezer) & (y IS-A refrigerator)
with the irrelevant features:
(x COLOR z) & (y COLOR z)

Figure 8. Version space with negative exceptions.

3.5 Improving the domain model

The existence of the rule's exceptions is due to the incompleteness of the domain model the language of which does not contain the expression distinguishing between the positive examples and the negative examples of the rule. But this is an excellent opportunity to develop the domain model by completing the descriptions of the existing concepts or even by defining new concepts. The definition of new concepts for the elimination of the rule's exceptions was called demand-driven concept formation by (Wrobel, 1989). In the following, we shall present two methods for the elimination of the negative exceptions of the rules. In what regards the positive exceptions, they are treated as positive examples of new rules to be learned.

3.5.1 Turning negative exceptions into negative examples by refining the descriptions of the known concepts

A negative exception of a rule may be transformed into a negative example by identifying (or defining) a new object feature that discriminates between the positive examples and the negative exception, as it is presented in Table 2.

Table 2. Refining object descriptions.

- Find a feature F of an object Oij from one positive example Ej such that:
  - F may be a feature of the corresponding objects Oi1,...,Oin, from ALL the positive examples;
  - F is not a feature of the corresponding objects O'ik,...,O'il from SOME of the negative exceptions Nk,...,Nl.
- Refine the descriptions of the objects Oi1,..., Oin by adding the feature F.
- Refine the descriptions of O'ik ,..., O'il by adding the feature NOT-F.
- Particularize the lower bound of the rule by adding the feature F (which is now shared by all the current positive examples)
- Remove Nk,...,Nl from the list of negative exceptions and add them to the list of negative examples.
- Repeat this procedure as long as such features could be found or could be defined by the user.

Let us consider the following negative exception from the version space in figure 8

(x IS-A dirty-cup1) & (y IS-A sink)

together with the positive examples

(x IS-A clean-cup1) & (y IS-A cabinet)

(x IS-A sugar) & (y IS-A sugar-box)

To transform this negative exception into a negative example, the system may analyze all the features of clean-cup1 in order to find one that may be a feature of sugar without being a feature of dirty-cup1. Or, it may analyze all the features of sugar in order to find one that may be a feature of clean-cup1 without being a feature of dirty-cup1. It may also look for a feature of cabinet or for a feature of sugar-box that is not a feature of sink.

In the domain model from figure 1 sugar-box has the feature (sugar-box IS closed), which is not a feature of cabinet and sink. Therefore, the system makes the hypothesis

(cabinet IS closed)

NOT(sink IS closed), rewritten as (sink IS opened)

and asks the user to validate them.

Let us suppose that the user validates these hypothesis. This means that (y IS closed) is a feature that discriminates between the known positive examples and the analyzed negative exception. By introducing the discriminating feature into the rule's conditions, the considered negative exception becomes a negative example (see figure 13).

Also, the descriptions of the sink and the cabinet are refined by adding the discovered properties:

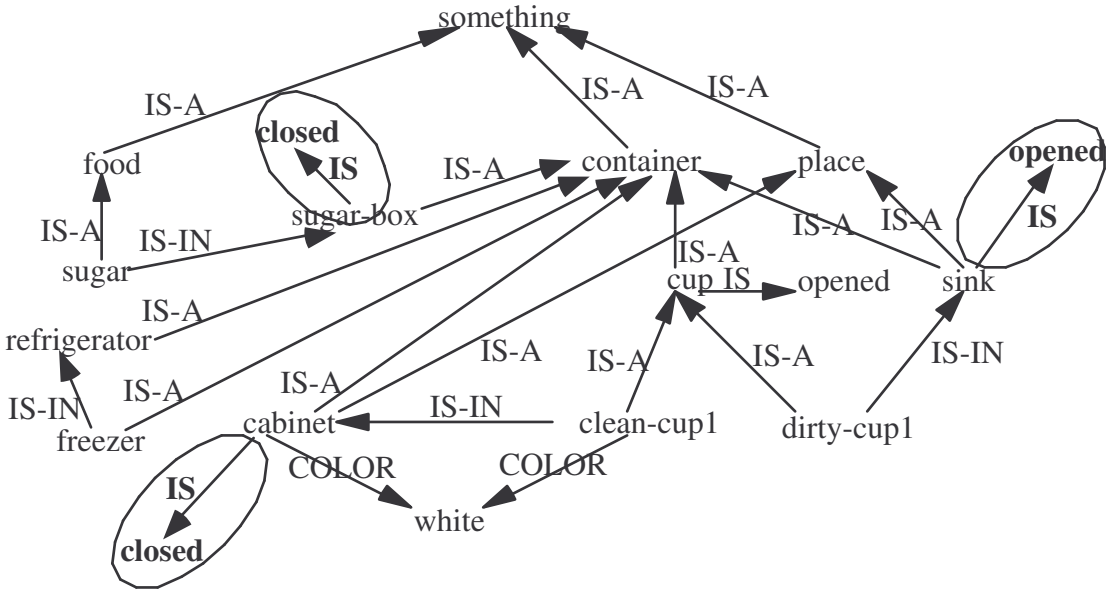

Figure 9. Goal-driven transfer of the property IS from sugar-box to cabinet and sink.

This is a case of goal-driven property transfer from one concept to another. It is quite often for an expert to define a feature of a concept but to forget to specify it when describing another concept. With the above presented method, the learner may discover and remove such an incompleteness.

3.5.2 Turning negative exceptions into negative examples by defining new concepts

Another method of transforming a negative exception of a rule into a negative example consists of defining a new concept that discriminates between the positive examples and the negative exception, as shown in Table 3.

Table 3. Definition of new concepts.

- Define a new concept C that
  - covers corresponding objects Oi1,...,Oin from ALL the positive examples;
  - does not cover the corresponding objects O'ik,...,O'il from SOME of the negative exceptions Nk,...,Nl;
  - is less general than the concept from the lower bound that covers Oi1,...,Oin;
  - may be given a meaningful name by the user.
- Replace with C the concepts from the upper bound and the lower bound that cover Oi1,...,Oin.
- Remove Nk,...,Nl from the list of the negative exceptions and add them to the list of the negative examples.
- Repeat this procedure as long as there are negative exceptions and meaningful concepts can be defined.

Let us consider the second negative exception from the version space in figure 8

(x IS-A freezer) & (y IS-A refrigerator)

together with the positive examples

(x IS-A clean-cup1) & (y IS-A cabinet)

(x IS-A sugar) & (y IS-A sugar-box)

One may define a concept covering clean-cup1 and sugar, not covering freezer, and being less general than something:

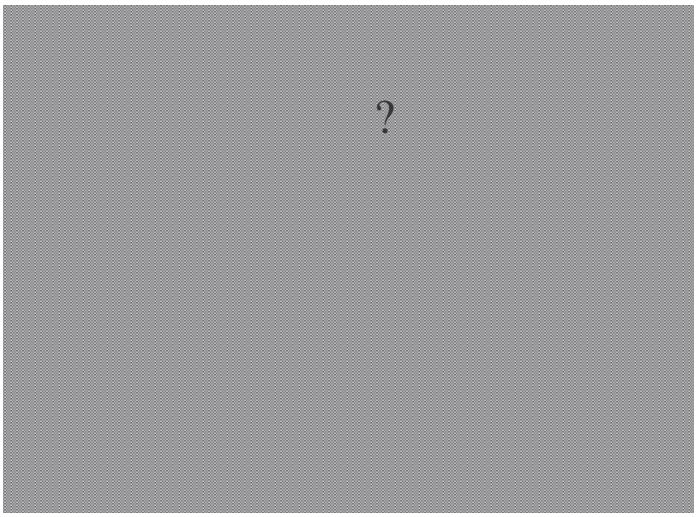

Figure 10. An object concept that would eliminate a negative exception.

One may also define a concept covering cabinet and sugar-box, not covering refrigerator, and being less general than container:

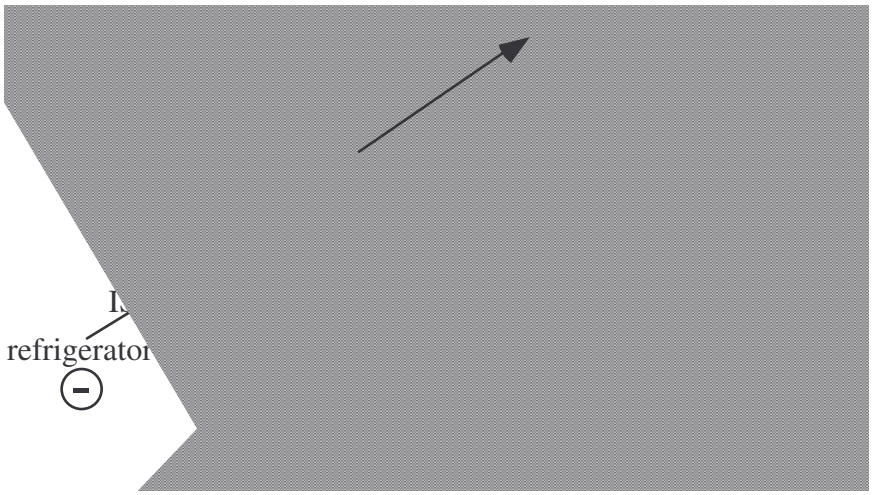

Figure 11. Another object concept that would eliminate a negative exception.

Analyzing the features of the covered and the uncovered objects (in the context of the problem solving episode from figure 2) the expert may realize, for instance, that sugar and clean-cup1 are objects that could be moved by the robot while freezer is not such an object. Therefore, he may name the corresponding intermediate concept movable-obj (i.e. object that could be moved by the robot) and may approve its introduction into the domain model. In such a case, however, the expert should also analyze the other concepts from the model in order to find the correct place for the new concept:

Figure 12. Definition of the concept movable-obj.

One may notice that this is a case of goal-driven conceptual clustering in which known concepts are clustered under newly defined ones, in order to improve the consistency of the learned rules.

By defining the 'movable-obj' concept the version space of the rule to be learned becomes the one shown in figure 13.

IF
upper bound
(x IS-A movable-obj) & (y IS-A something) & (x IS-IN y) & (y IS closed)
lower bound
(x IS-A movable-obj) & (y IS-A container) & (x IS-IN y) & (y IS closed)
THEN
the problem
TAKE x
has the solution
OPEN y
TAKE x FROM y
with the positive examples:
(x IS-A clean-cup1) & (y IS-A cabinet)
(x IS-A sugar) & (y IS-A sugar-box)
with the negative examples:
(x IS-A dirty-cup1) & (y IS-A sink)
(x IS-A freezer) & (y IS-A refrigerator)
with the irrelevant features:
(x COLOR z) & (y COLOR z)

Figure 13. The version space after turning the negative exceptions into negative examples.

A similar method is used by BLIP (Wrobel, 1989). The main difference is that BLIP always defines only one concept that discriminates between the objects from the positive examples and the corresponding objects from the exceptions of a rule. Our method may define several concepts, each eliminating at least one exception, if they are meaningful concepts in the modelled domain. If we want the domain model to approximate the real world as close as possible, there is no reason to believe that one has always to define only one concept for eliminating all the exceptions. Moreover, it may not be always desirable to eliminate all the exceptions because this may result in a complicated domain model.

3.6 The use of the action models

In the above sections we have supposed that the system does not have any models of the actions involved in the initial problem solving episode. However, if the system has even incompletely learned action models then the learning of the new rule is speeded up and at the same time with learning the rule the system is also improving the action models.

Let us suppose, for instance, that the system disposes of the incompletely learned action models from figure 14. By using these action models the system may build the tree in figure 15 which "proves" that the sequence of actions

OPEN cabinet, TAKE clean-cup1 FROM cabinet

achieves the goal of the action

TAKE clean-cup1.

The tree in figure 15 is, however, only a plausible proof. Indeed, to build this tree, the system used the upper bounds of the applicability conditions of the action models from figure 14, upper bounds that

may cover situations in which the corresponding actions are not applicable. Also, the system had to abduct the fact (cabinet IS closed), which is not present in the domain model from figure 1. Therefore this tree has to be validated by the user.

IF
upper bound
(x IS-A something)
lower bound
(x IS-A bottle)
THEN
the action
TAKE x
has the effect
(Roby HAS x)
with the positive examples
(x IS-A wine-bottle1)
(x IS-A whisky-bottle1)

IF
upper bound
(x IS-A something) &(y IS-A something) &
(x IS-IN y) & (y IS opened)
lower bound
(x IS-A fruit) & (y IS-A fruit-basket) &
(x IS-IN y) & (y IS opened)
THEN
the action
TAKE x FROM y
has the effects
(Roby HAS x)
NOT(x IS-IN y)
with the positive examples
(x IS-A apple) & (y IS-A fruit-basket)
(x IS-A banana) & (y IS-A fruit-basket)

IF
upper bound
(x IS-A something) & (x IS closed)
lower bound
(x IS-A bottle) & (x IS closed)
THEN
the action
OPEN x
has the effects
(x IS opened)
NOT(x IS closed)
with the positive example
(x IS-A wine-bottle1)
(x IS-A whisky-bottle1)

Figure 14. Incompletely learned action models.

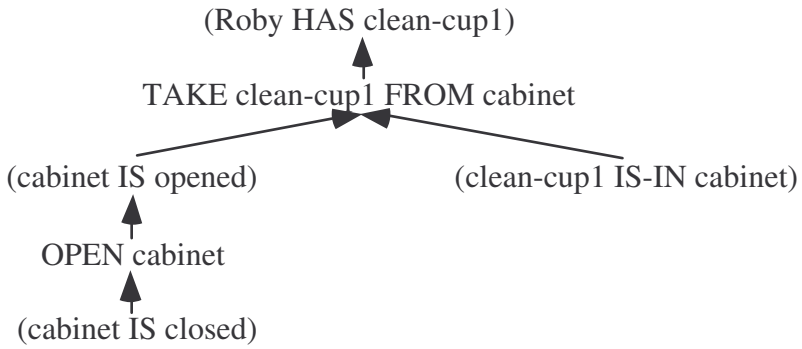

Figure 15. A plausible proof of the correctness of Example1 in figure 2.

The validation of the plausible proof from figure 15 has the following important consequences:
- the leaves of the tree represent the explanation of the problem solving episode in figure 2

(cabinet IS closed) & (clean-cup1 IS-IN cabinet)

- the abducted fact (cabinet IS closed) is introduced into the domain model;
- each action instance from the tree represents a new positive example for the corresponding action model that could be generalized to cover it. For instance, the lower bound of the precondition of 'OPEN x' is generalized to cover (x IS-A cabinet) & (x IS closed):

IF
upper bound
(x IS-A something) & (x IS closed)
lower bound
(x IS-A container) & (x IS closed)
THEN
the action
OPEN x
has the effects
(x IS opened)
NOT(x IS closed)
with the positive examples
(x IS-A wine-bottle1)
(x IS-A whisky-bottle1)
(x IS-A cabinet)

Figure 16. Improving the model of the action OPEN.

The action models may also be used to generalize the plausible proof in figure 15 (Tecuci and Kodratoff, 1990):

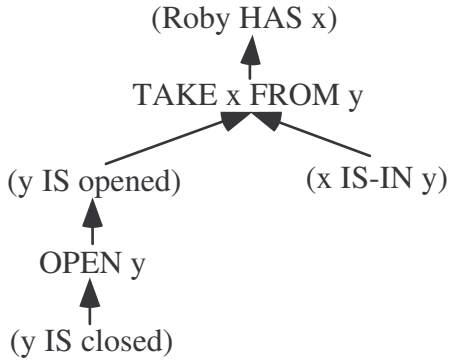

Figure 17. Generalization of the plausible proof in figure 15.

The leaves of this tree represent the analogy criterion to be used for generating new problem solving episodes: (y IS closed) & (x IS-IN y).

Therefore, the system is able to define the initial version space shown in figure 18 and may search the rule in this space by following the steps indicated in Table 1.

IF
upper bound
(x IS-A something) & (y IS-A something) &
(x IS-IN y) & (y IS closed)
lower bound
(x IS-A clean-cup1) & (y IS-A cabinet) &
(x IS-IN y) & (y IS closed)
THEN
the problem
TAKE x
has the solution
OPEN y
TAKE x FROM y
with the positive example:
(x IS-A clean-cup1) & (y IS-A cabinet)

Figure 18. A version space defined with the help of incomplete action models.

If the domain model contains complete action models then the method presented in Table 1 becomes explanation based learning. In such a case, the generalized tree from figure 17 would be a logical proof and the leaves of this tree would be the condition of the rule in figure 3 (Tecuci and Kodratoff, 1990).

4. Conclusions

We have presented a methodology for domain modeling and knowledge acquisition that involves a synergistic combination of different learning strategies. This methodology was implemented in Common Lisp on Macintosh. After using it to build experimental domain models in several domains (manufacturing, geography, chemistry, etc.) we have concluded that it has a high potential for partially automating the process of building expert systems.

There are several directions of improvement and development of this methodology. One direction consists of developing methods for automating the construction of the initial domain model. A interesting approach is provided by the BLIP and MOBAL systems (Morik, 1989; Wrobel, 1989) which are able to build such an initial domain model from user provided facts. Another direction of research concerns the development of methods for restructuring the domain model. The system should be able not only to define new concepts and to add new features at the known concepts but also to delete concepts or to remove features from the known concepts, managing all the consequences of such changes in the domain model. Also the multistrategy rule learning method may be improved by better

integrating the existent learning strategies and by adding new ones, in order to increase the learning capabilities of the system. This research direction is closely related to an on going effort at the Center for Artificial Intelligence to define a unifying theory of machine learning and a general multistrategy task-adaptive learning methodology based on this theory (Michalski, 1990; Tecuci and Michalski, 1990).

Acknowledgements

The author thanks Yves Kodratoff, Ryszard Michalski and Katharina Morik for useful discussions that influenced this work, Michael Hieb, Ken Kaufman and the referees for useful comments and criticisms, and Janet Holmes for help in the preparation of this report.

This research was partly done in the Artificial Intelligence Center of George Mason University. Research activities of the Center are supported in part by the Defence Advanced Research Projects Agency under grant No. N00014-87-K-0874, administrated by the Office of Naval Research, and in part by the Office of Naval Research under grants No. N00014-88-K-0226 and No. N00014-88-K-0397.

The initial research was done under the support from the Romanian Academy, the Research Institute for Informatics in Bucharest and the French National Research Center.

References

Bhatnagar, R.K., and Kanal L.N. (1986) Handling Uncertain Information: A Review of Numeric and Non-numeric Methods, in Kanal L.N. and Lemmer J.F. (eds) *Uncertainty in Artificial Intelligence*, Elsevier Science Publishers, North-Holland, 3-26.

Boose, J.H., Gaines, B.R., and Ganascia, J.G.(eds), *Proceedings of the Third European Workshop on Knowledge Acquisition for Knowledge-based Systems,* Paris, July, 1989.

Danyluk, A.P., The Use of Explanations for Similarity-Based Learning, *Proceedings of IJCAI-87*, pp. 274-276, Milan, Italy, 1987.

Dietterich, T.G., and Flann, N.S., An Inductive Approach to Solving the Imperfect Theory Problem, *Proceedings of 1988 Symposium on Explanation-Based Learning,* pp. 42-46, Stanford University, 1988.

DeJong G., and Mooney R., Explanation-Based Learning: An Alternative View, in *Machine Learning,* vol.1, no. 2, pp. 145-176, 1986.

Kodratoff Y., and 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 Y., and 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, Y., and Michalski, R.S. (eds), *Machine Learning: An Artificial Intelligence Approach,* Morgan Kaufmann, vol.III, 1990.

Lebowitz, M., Integrated Learning: Controlling Explanation, *Cognitive Science,* Vol. 10, No. 2, pp. 219-240, 1986.

Michalski, R.S., Carbonell J.G., and Mitchell T.M. (eds), *Machine Learning: An Artificial Intelligence Approach,* Morgan Kaufmann, vol.I, 1983, vol.II, 1986.

Michalski R.S., Theory and Methodology of Inductive Learning, *Readings in Machine Learning*, Dietterich T., and Shavlik J. (eds.) Morgan Kaufmann 1990.

Michalski R. S., Toward a Unified Theory of Learning: Multistrategy Task-adaptive Learning, Submitted for publication in *Machine Learning Journal*, 1990.

Minton, S., Carbonell, J.G., Etzioni, O., Knoblock C., Kuokka D.R., Acquiring Effective Search Control Rules: Explanation-Based Learning in the PRODIGY System, *Proceedings of the 4th International Machine Learning Workshop,* pp. 122-133, University of California, Irvine, 1987.

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

Mitchell T.M., Keller R.M., and Kedar-Cabelli S.T., Explanation-Based Generalization: A Unifying View, *Machine Learning,* vol.1, no.1, pp. 47-80, 1986.

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

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

Porter B., & Mooney R. (eds), *Proceedings of the Seventh International Workshop on Machine Learning*, Texas, Austin, 1990, Morgan Kaufman.

Segre, A.M. (ed.), *Proceedings of the Sixth International Workshop on Machine Learning*, Cornell University, Ithaca, New York, June 26-27, 1989.

Tecuci G., Kodratoff Y., Bodnaru Z., and Brunet T., 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 G., DISCIPLE: A Theory, Methodology, and System for Learning Expert Knowledge, Ph.D. Thesis, University of Paris-Sud, 1988.

Tecuci, G. and Kodratoff Y., Apprenticeship Learning in Imperfect Theory Domains, in Kodratoff Y., and Michalski R.S. (eds), *Machine Learning: An Artificial Intelligence Approach,* vol. III, Morgan Kaufmann, 1990.

Tecuci, G. and Michalski R., A Method for Multistrategy Task-Adaptive Learning Based on Plausible Justification, to appear in *Reports of Machine Learning and Inference Laboratory,* George Mason University, 1991.

van Melle, W., Scott, A.C., Bennett, J.S., and Peairs, M., The EMYCIN Manual, Report no. HPP-81-16, Computer Science Department, Stanford University, 1981.

Zhang, J. Learning Flexible Concepts from Examples: Employing the Ideas of Two-Tiered Concept Representation, PhD Thesis, University of Illinois at Urbana-Champaign, 1990.

Wilkins, D.C., Clancey, W.J., and Buchanan, B.G., *An Overview of the Odysseus Learning Apprentice*, Kluwer Academic Press, New York, NY, 1986.

Wrobel S., Demand-Driven Concept Formation, in Morik K.(ed), *Knowledge Representation and Organization in Machine Learning*, Springer Verlag, Berlin 1989.