Mental Models and Human-Computer Interaction 2
M.J. Tauber and D. Ackermann
Elsevier Science Publishers B.V. (North-Holland), 1991

THE PSYCHOLOGICAL PROCESSES OF CONSTRUCTING A MENTAL MODEL WHEN LEARNING BY BEING TOLD, FROM EXAMPLES, AND BY EXPLORATION

Franz Schmalhofer and Otto Kühn

Deutsches Forschungszentrum für Künstliche Intelligenz, Kaiserslautern, FRG

ABSTRACT

The current paper presents a detailed description of an integrative simulation model, which specifies the psychological knowledge acquisition process in learning by being told (learning from text), in learning from examples (learning from a demonstration) and in learning by exploration. A learner's prior domain knowledge, which may consist of domain specific knowledge and general heuristics, has a critical influence in what is being learned. For various degrees of prior knowledge, the simulation specifies the different ways by which a mental model may be formed. Consequently, it can be applied for evaluating the effectiveness of the different learning methods and instruction materials as well as the influence of a learner's prior knowledge.

Introduction

It has become well recognized that in order to establish user-oriented design principles, one indeed needs to know which knowledge users may employ to successfully work with computer systems and how such knowledge can be acquired. In previous research, formal models have been developed which specify the procedural knowledge that is sufficient for performing various tasks. Such knowledge which has been termed how-to-do-it knowledge is formally represented by production rules.

The Cognitive Complexity Theory (CCT) of Kieras and Polson (1985) belongs to this class of models. The CCT assumes that the time required for learning to perform some task can be predicted by the number of production rules which must be newly learned. This prediction has been experimentally confirmed in several laboratory experiments (e.g. Polson, Muncher and Engelbeck, 1986; Ziegler, Hoppe and Fähnrich, 1986). Typically, the subjects of these experiments are explicitly trained step by step in performing the various unit tasks (Card, Moran and Newell, 1983), such as deleting a word.

For example, in the experiment by Polson et al. subjects were to anticipate the next operating step in performing a task. After they failed twice they were explicitly told the operation which they had to perform. Subjects were thus obliged to memorize a sequence of operations. The experimental results demonstrate that under these conditions how-to-do-it knowledge that can be represented by production rules is directly acquired. In particular, the mean time to master a task at a given position in a training sequence is determined by the number of production rules which must be newly learned for that task.

In these experiments subjects were quite restricted in what and how they learned. At each step in the training task subjects had to learn exactly one operation. Other than the sequential order no relation was established among these steps of the training procedure. Basically, the training procedure consisted in the memorization of a sequence of operations. Due to this training procedure learners could hardly achieve an understanding of how the system works. They could not utilize their prior knowledge to derive a deeper understanding of the effectiveness of their actions. Since such a learning procedure is rather untypical, it is questionable to what extent the CCT should be used for deriving user-centered design principles.

In a different line of research, it has been argued that a user's mental model of a system is of critical importance (Norman, 1983; Halasz and Moran, 1983). Contrary to the how-to-do-it knowledge a mental model represents the how-it-works knowledge. A mental model is a representation of the internal structures and processes of a system at some uniform level of description. From a mental model any operating procedure for a device may supposedly be inferred by general reasoning processes. Therefore acquiring a mental model from which all the relevant how-to-do-it knowledge can be derived may be overall more parsimonious than memorizing a collection of operating procedures.

For building or updating a mental model of some system, humans may employ a combination of different learning methods. Users may study a text, learn from a demonstration of specific interactions with the system (learning from examples) or simply explore the system by themselves. Rather than specifying only the resulting products, the information processing of the different learning methods themselves should be modeled. Since a learner's prior domain knowledge often determines what is being learned from an instructional episode, the prior knowledge must be explicitly represented in such a learning model. By specifying different amounts and kinds of prior knowledge individual differences between users can be taken into account when deriving learnability predictions.

In the present paper, we will present such a computer simulation which describes how a mental model is formed and updated by a combination of learning methods:

1) by being told (text learning),
2) from a demonstration (examples) and
3) by exploration.

The various kinds of prior knowledge which are utilized during this learning process will be delineated. In addition it will be shown how how-to-do-it knowledge can be derived from such a mental model. Expertise differences in the construction of a mental model will be accounted for by differences in the prior domain knowledge whereas the learning strategies remain unchanged.

The construction of a mental model

General processing goals

Under most circumstances learning is goal driven. The goal of knowledge acquisition (KA-goal) is to form or update a mental model which will be well suited for deriving how-to-do-it knowledge when specific tasks arise. In order to allow for such derivations, knowledge which is only implicitly contained in the mental model must be made explicit to a certain degree. Without this knowledge explication in the knowledge acquisition phase the processing demands for performing specific tasks later could not be met. The goal of learning therefore is to construct a mental model so that various tasks can be solved relatively easily. Learning can thus be seen as a problem solving process in which knowledge is explicated so that some conjectured tasks can be solved in the future.

From the general learning goal four more specific processing goals can be derived: explicitness, parsimony, coherence, and consistency. The constructed mental model should contain as much explicit knowledge as possible. Because of memory limitations the knowledge should also be represented parsimoniously. Explicitness and parsimony are contradicting processing goals. Therefore a compromise between these processing goals must be achieved. Since humans are better able to remember coherent information, the new knowledge should be related to the prior existing knowledge. In addition, the mental model should be consistent, since it should not provide contradictory solutions for the same task.

In learning by being told, general information is usually presented to the learner. In order for this information to be useful, the KA-goal requires that more specific information be derived from it. The supposedly relevant knowledge which is implied by the general statements must be made explicit. Examples, on the other hand, present very specific information. In order for an example to be useful for solving other tasks, the KA-goal requires that more general information be inferred from it. Examples must be generalized so that the resulting knowledge is applicable for a larger variety of tasks. Thus two different subgoals, one for learning from text and one for learning from examples, result from the general KA-goal. The KA-goal thus leads to a mental model which contains knowledge that is rather explicit and at the same time general enough to be applied to a number of tasks.

Based on the four general processing goals of knowledge acquisition, an integrative computer simulation of human learning was developed. Since learning is driven by the existing prior knowledge, the representation of (prior) knowledge will be discussed before the different learning methods are described.

Representation of knowledge

Three logically different types of knowledge are distinguished:

1) known rules and facts,

2) hypotheses, and

3) heuristics

which can generate hypotheses. The three following forms are used to represent knowledge (please note that capital letters at the beginning of a word identify variables):

1) *known(Domain,FactOrRule,Info)*
2) *hyp(Domain,FactOrRule,Info)*
3) *heur(Domain,FactOrRule,Info).*

Rules and facts which are known to be true are represented by form 1). Hypotheses which were inductively formed and may therefore be incorrect are represented by form 2). Form 3) is used to specify heuristics which are applied for generating and modifying hypotheses.

The knowledge base is segmented into *Domains* which may be accessed selectively. *FactOrRule* specifies the knowledge contents. Info provides additional information about a knowledge element such as confidence and usefulness counters, and how the particular knowledge element has been obtained.

Domains are hierarchically organized. Although the proposed simulation model is general and not tied to a specific subject domain it will be presented for the acquisition of basic LISP knowledge. For the learning of elementary LISP the relevant knowledge is structured according to the following hierarchy:

$$\begin{tikzcd}
 & \textit{commonsense} \arrow[d, dash] & \\
 & \textit{mathematics} \arrow[d, dash] & \\
 & \textit{lisp} \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\
\textit{list} & \textit{first} & \textit{rest} \ldots
\end{tikzcd}$$

Together the knowledge domains *lisp, list, first, rest,..* form the user's mental model of the LISP system. When such a mental model is being built up, knowledge from related domains is utilized to derive inferences and to form hypotheses.

Description of the relevant prior knowledge

In the case of LISP knowledge the domains of mathematics and commonsense are assumed to form the relevant prior knowledge. Commonsense knowledge is only accessed, if the subordinate, more specific knowledge is insufficient for accomplishing the KA-goal.

It consists of the ordinary and universal knowledge, such as how to compare statements to empirical observations, the transitivity of subclass relations and counting. In addition the commonsense knowledge contains five types of general heuristics which can generate or modify hypotheses in a domain: The empirical generation heuristics produce hypotheses about specific observations being true in general. The wishful thinking heuristics generate hypotheses so that some path of reasoning can be completed without any further consideration. Generalisation and differentiation heuristics, on the other hand, modify already existing hypotheses. Selection heuristics select a specific element from some class under certain constraints. For example, alternative hypotheses would be generated by selection heuristics.

Some examples of commonsense knowledge will be presented:

One piece of commonsense knowledge says that *Something* is concluded to be correct, if the requirements for the activated attribute of the relevant concept are satisfied. More explicitly, *Something* satisfies an attribute requirement of a concept correctly, if

1. the concept requires the attribute to have a certain value (*RequiredValue*) and
2. an analysis of that *Something* which is driven by the attribute of the concept yields a *GivenValue* and
3. the *GivenValue* corresponds to the *RequiredValue*.

Its formal representation in PROLOG is:

```
correct(Attribute,Concept,Something) :-
  required(Attribute,Concept,RequiredValue),
  given(Attribute,Something,GivenValue),
  correspond(RequiredValue,GivenValue).
```

If no hypothesis exists about the *RequiredValue* of the attribute, the following empirical generation heuristic is used. It generates the hypothesis that the *GivenValue* is also the *RequiredValue* for the attribute.

```
correct(Attribute,Concept,Something) :-
  not hyp(required(Attribute,Concept,RequiredValue)),
  given(Attribute,Something,GivenValue),
  generate_hyp(required(Attribute,Concept,GivenValue)).
```

If a hypothesis contradicts a given value of an attribute the following generalization heuristic may be employed. For example, for the attribute *number_of_arguments* a *RequiredValue* of 1 and *GivenValue* of 3 may yield the generalization *RequiredValue* of *at_least(1)*.

```
correct(Attribute,Concept,Something) :-
  hyp(required(Attribute,Concept,RequiredValue)),
  given(Attribute,Something,GivenValue),
  find_general(RequiredValue,GivenValue,Generalization),
  generalize_hyp(required(Attribute,Concept,GivenValue),
    required(Attribute,Concept,Generalization)).
```

The specialization heuristic works accordingly.

To the domain of mathematics belongs the knowledge about functions in general (i.e. a function schema with the slots: number of arguments, type of arguments, and input/output relation). It is represented by:

```
correct_function(Fname,Arguments,Result) :-
  correct(number_of_arguments,Fname,Arguments),
  correct(type_of_arguments,Fname,Arguments),
  correct(io_relation,Fname,(Arguments,Result)).
```

This knowledge from the domains commonsense and mathematics is used together with the instructional materials to form a basis for the mental model of the LISP system. The basic distinctions of the mental model are assumed to be acquired through learning by being told. This learning method will be described later. From an instructional text about the definitions of atoms and lists and the evaluation of an input by the LISP system an initial (and incomplete) mental model of the LISP system is obtained.

The initial mental model

The knowledge about LISP contains information about structures and processes. The structures describe LISP objects and various relations among them. Their representation depends upon the instructional material. For example, the instructional material determines which of the following two representations is constructed for lists:

```
is_instance([],list).
is_instance([X|Xs],list) :-
   is_instance(X,s_expr),
   is_instance(Xs,list).

is_instance(X,list) :-
  begins_with(X,'['),
  ends_with(X,']'),
  contains(X,Xs),
  are_instance(Xs,s_expr).
```

Atoms and lists are subclasses of s-expressions, which is simply represented by:

```
isa(atom,s_expr).
isa(list,s_expr).
```

The evaluation of LISP objects is represented by various process descriptions. For example, the evaluation of a function call as input to the LISP system is represented by:

```
eval(Input,Result) :-
    funcall(Input,Fname,ArgSpecs),
    eval_argspecs(ArgSpecs,Arguments),
    correct_function(Fname,Arguments,Result).
```

Supposedly, the acquisition of additional LISP knowledge (e.g. about specific functions such as FIRST, REST, ...) depends upon how well the initial mental model has been formed. Different learning methods may be employed. A particular LISP-function like the function FIRST can be learned in three different ways.

1) It can be learned from the following sentences:

   FIRST is a LISP function.
   It takes exactly one argument.
   The argument must be a list.
   The function FIRST returns the first element of the argument.

2) Alternatively, the function FIRST can be learned from a sequence of examples such as:

   ```
   (FIRST '(A B)) --> A
   (FIRST 'A) --> ERROR
   ```

3) A third way of learning FIRST is to present one (simple) example (or just the name of the function) as a start for exploring that function by interacting with the LISP interpreter.

Usually a combination of these learning methods will be used when building a mental model.

Learning by being told

The previously listed sentences defining the LISP-function FIRST are represented by the following clauses:

```
lisp_function(first).
required(number_of_arguments,first,1).
required(type_of_arguments,first,list).
required(io_relation,first,the_first_element_of_argument).
```

These clauses are sequentially integrated into the initial mental model. This process is driven by the KA-goal. At first, it is examined whether a clause is already explicitly or implicitly known (redundant clauses) or whether it contradicts already existing knowledge (contradicting clauses).

Redundant clauses

A clause which is redundant with respect to knowledge of type known is only added to the mental model, if it enhances the explicitness of the mental model, i.e. if a long inference chain would otherwise be needed to derive it. A clause which is redundant with respect to some hypothesis replaces the respective hypothesis. No further processing is required for redundant clauses in order to achieve the learning goal.

Contradicting clauses

For contradicting clauses, the contradiction with the mental model must be resolved. If the clause contradicts information of type *known*, the learning process is interrupted and a question is asked to the teacher (see Learning by exploration). If the clause contradicts a hypothesis (information of type *hyp*), three different cases must be distinguished:

1) If the contradicting clause is more general than the hypothesis, the hypothesis is substituted by the clause, and any inferences derived from the hypothesis are generalized accordingly.
2) If the contradicting clause is more specific, the hypothesis is too general. The overgeneralized hypothesis is consequently replaced by the contradicting clause (which is of type *known*). The clauses

which have been derived from the overgeneralized hypothesis are specialized accordingly. For instance, the hypothesis required(type_of_arguments,first,s_epxr) would be replaced by the told fact required(type_of_arguments,first,list), and the derived clause

```
interaction([first,$ X],Y) :-
    is_instance(X,s_expr),
    the_first_part_of(X,Y)
```

would be specialized to

```
interaction([first,$ X],Y) :-
    is_instance(X,list),
    the_first_part_of(X,Y).
```

3) If the clause contradicts a hypothesis without being more general or more specific, the hypothesis is replaced by the contradicting clause, and any information derived from the hypothesis is withdrawn from the mental model.

New clauses

New clauses (i.e. clauses which are neither redundant nor contradicting) become part of the mental model. The coherence of the new clause to the mental model is established by deriving mostly forward inferences from the new clause and the mental model. Forward inferences are tried first, since texts usually present general information before the more specific information.

When generating inferences the knowledge is searched from the specific to more general domains. Superordinate domains such as mathematics and commonsense may thus be used for constructing the mental model. Since the most recently acquired knowledge is always stored at the top of each domain, recently acquired knowledge is preferably utilized. By storing inferences in the mental model, its explicitness is increased. Since the obtained inference can be used to find another inference (and so on) the explicitness of the knowledge base can be enhanced even further. The obtained forward inferences can be simplified by applying backward inferences.

A large (possibly infinite) number of inferences could be derived from each statement. Because of the parsimony criterion the inferencing process has been restricted. Only one forward inference is generated at each level and only i inferences are allowed in total. Thereby a human's limited processing resources are taken into consideration. i is assumed to be a parameter of the model.

In the following it is shown, how the new clause *lisp_function(first)* is integrated into the initial mental model. At first, the forward inference:

```
funcall(Input) :-
    Input = [first | ArgSpecs].
```

is obtained from the new clause and the following clause of the initial mental model:

```
funcall(Input) :-
    Input = [Fname | ArgSpecs],
    lisp_function(Fname).
```

This forward inference is generated by unifying (the head of) the new clause, *lisp_function(first)*, with a condition of a known clause, *lisp_function(Fname)*. The unifying condition is dropped from the clause, and the variable bindings established in the unification are maintained.

In the second step the inference:

```
eval(Input,Result) :-
    Input = [first | Argspecs],
    eval_argspecs(Argspecs,Arguments),
    correct_function(first,Arguments,Result).
```

is obtained from the inference derived in the first step and the following clause:

```
eval(Input,Result) :-
    funcall(Input,Fname,ArgSpecs),
    eval_argspecs(ArgSpecs,Arguments),
    correct_function(Fname,Arguments,Result).
```

Due to the obtained inferences less processing is required, when subsequent examples are studied or have to be verified, or when a programming task has to be solved. A usefulness value of 1 is stored with each inference. This value is incremented each time the inference is used. After a certain time period, inferences with low usefulness values are deleted from the knowledge base.

Learning from examples

The initial mental model may also be elaborated by studying specific examples of interactions with the LISP system, such as:

```
(LIST 'A 'B)--> (A B)
(LIST '(A B) 'C '(D E))-->((A B) C (D E))
(LIST A B) --> ERROR
```

The first two inputs to the LISP system can actually be evaluated. Therefore they are called positive example interactions. Inputs which yield an error message like the third example are called negative examples. These three examples may be represented by:

```
interaction([list, $ a, $ b], [a,b]).
interaction([list, $ [a,b], $ c, $ [d,e]], [[a,b],c,[d,e]]).
interaction([list, a, b], error).
```

Again these examples are sequentially integrated into the mental model so that the learning goal is achieved. For each example it is determined whether it is consistent with the already existing mental model. This is accomplished by utilizing the mental model to mentally evaluate the input of the example. Because of the assumed human processing limitations, again only j inferencing steps are performed. If this processing limit is reached, the existing mental model is not explicit enough for evaluating the example input. Consequently, no further processing of the example would be performed. When the next example is studied, the previously achieved inferences are supposedly primed so that overall more than j inferencing steps can be performed. For a more explicit mental model the processing demands will be below the processing limits and the mental evaluation may yield a result. If the mental evaluation yields the same result as specified in the example, the example is called redundant with respect to the already existing mental model. If the mental evaluation yields a different result than specified in the example the example contradicts the existing mental model. If the mental model does not contain sufficient information for evaluating the example input, the example is called new.

Redundant examples

Although redundant examples do not contain any novel information they may still be used to explicate knowledge which is so far only implicit in the mental model. During the mental evaluation of an example input the mental model was presumably used to construct an explanation how the input to the LISP system is transformed into the output. From this explanation a schema or template is constructed. Such templates make the mental model more explicit and can be used to evaluate inputs to the LISP system in a small number of processing steps.

For the first example the following explanation may be obtained from an explicit mental model:

```
interaction([list, $ a, $ b], [a,b]) since
  funcall([list, $ a, $ b], list, [$ a, $ b]) and
  eval_argspecs([$ a, $ b], [a,b]) and
  correct_function(list, [a,b], [a,b]).
  funcall([list, $ a, $ b], list, [$ a, $ b]) since
    [list, $ a, $ b] = [list, $ a, $ b],
    lisp_function(list).
  eval_argspecs([$ a, $ b], [a,b]) since
    eval($ a, a) and
    eval($ b, b).
  correct_function(list, [a,b], [a,b]) since
    correct(number_of_arguments, list, [a,b]) and
    correct(io_relation, list, ([a,b],[a,b])).
    correct(number_of_arguments, list, [a,b]) since
      required(number_of_arguments, list, any) and
      given(number_of_arguments, [a,b], 2) and
      correspond(any,2).
    correct(type_of_arguments, list, [a,b]) since
      required(type_of_arguments, list, s_expr) and
      is_instance(a, s_expr) and
      is_instance(b, s_expr).
    correct(io_relation, list, ([a,b],[a,b])) since
      required(io_relation, list, list_of_arguments) and
      given(io_relation, ([a,b],[a,b]),
list_of_arguments).
```

From this lengthy explanation the following more concise template is constructed:

```
interaction([list, $ X, $ Y], [X,Y]) :-
    is_instance(X, s_expr),
    is_instance(Y, s_expr).
```

This template is added to the mental model and increases its explicitness. As a consequence, several future LISP inputs will be evaluated more easily. Technically, explanation-based generalisation (Mitchell, Keller and Kedar-Cabelli, 1986) is used for constructing those templates.

Contradicting examples

An example is contradicting, if for its input a different result is predicted from the stored knowledge than is specified in the example. A contradicting example may only be observed if hypotheses were used for mentally evaluating the input. Since the examples themselves are known to be actual interactions with the LISP system, contradicting examples provide evidence that one or several hypotheses of the mental model are incorrect. Consequently, commonsense heuristics are employed to modify or even replace hypotheses so that the example can be explained. In addition, the inferences which were derived on the basis of the old hypotheses are appropriately modified or excluded.

New examples

For new examples, the heuristic rules from the commonsense knowledge are employed to generate hypotheses so that the example input is mentally evaluated into the observed example result. These hypotheses thus fill the gaps in the mental model and represent the new knowledge which was acquired from the example, the mental model, and commonsense heuristics. The mental evaluation including the generated hypotheses provides again an explanation how the specific example input is transformed into the example result. From the obtained explanation a template is constructed. The template which also includes the relevant hypotheses is then added to the mental model.

Assume that only the initial mental model had been constructed, and the example *(LIST 'A 'B) --> (A B)* is studied. Then the commonsense heuristics may generate the following four hypotheses:

```
1) hyp(list,lisp_function(list),(1,[],[])).
2) hyp(list,required(number_of_arguments,list,2),(1,[],[])).
3) hyp(list,required(type_of_arguments,list,[atom,atom]),(1,[],[])).
4) hyp(list,required(io_relation,list,list_of_arguments),(1,[],[])).
```

Hypothesis 1) was generated by the wishful thinking heuristic, and hypotheses 2), 3), and 4) were generated by the empirical generation heuristic. A confidence value of 1 is initially stored with each hypothesis. When a hypothesis is reused INFO is updated as described by Schmalhofer (1986).

With these hypotheses, the second example cannot be explained. Consequently, hypotheses 2 and 3 are modified by the generalisation heuristics into:

2) *hyp(list,required(number_of_arguments,list,at_least(2)),(2,[1],[])).*
3) *hyp(list,required(type_of_arguments,list,s_expr),(1,[atom],[])).*

Learning from negative examples

For a negative example it is also examined whether the example is consistent with the already existing mental model. In particular, it must be explained why the input cannot be evaluated. It must be shown that at least one of the necessary conditions for a correct input is violated. After such a condition is found, the error is attributed to the violation of this condition. If the violated condition is not of type known but only a hypothesis, its confidence value is increased, because the example provides evidence that the hypothesis is not overgeneralized. If no violated condition is found, one of the used hypotheses must be overgeneralized. The commonsense specialization heuristic is used to modify the hypothesis so that the error result can be explained.

For instance, if the hypothesis *required(type_of_arguments, first, s_expr)* had been generated and the example *interaction([first, $ a], error)* were studied, the hypothesis would be specialized to *required(type_of_arguments, first,list)*. Because no new templates are constructed, negative examples require less processing than positive examples.

Learning by exploration

In learning by exploration the learners themselves must instigate the learning process by asking a question to a teacher or to an environment. Whereas most efficient learning can take place when both general and specific questions may be asked, general questions are not allowed in learning by exploration. When exploring a computer system the type of questions that may be asked is even further restricted. For instance, the only type of questions a LISP-system can answer are "What is the result of evaluating a particular input?".

A model of learning by exploration must specify which questions can be asked by a learner with a given mental model, and how the answers to these questions will be used to update the mental model.

Learning by exploration can be used to test existing hypotheses as well as to generate and test new hypotheses.

Testing hypotheses through exploration

The mental model is searched for some insufficiently tested hypothesis. Insufficiently tested hypotheses are identified by low confidence values and by the lack of tested alternative hypotheses. From the selected hypothesis a question (i.e. a particular input to the LISP system) is then generated by a sequence of forward and backward inferences. The number of inferencing steps required, depends on the explicitness of the mental model. If templates are available, they are utilized for generating a question, and hence only a small number of inferences have to be performed. The generated question is then asked to the environment, and an answer is obtained. The question and the answer to the question constitute an example, from which knowledge can be acquired similarly to learning from examples.

For testing a hypothesis two different strategies can be employed: a confirmation strategy or a falsification strategy. Supposedly, the confirmation strategy is used until the tested hypothesis has a confidence value of k where k is a parameter of the model. Afterwards the falsification strategy is employed. When testing a hypothesis, human learners will usually have some expectations about the outcome of the test. The knowledge that is acquired from a particular interaction will depend both on the interaction itself as well as the subjects' expectations and the strategy of hypothesis testing employed.

How the mental model may be elaborated through testing hypotheses is best illustrated by an example. Assume, the example:

```
(SET 'L '(A B)) --> (A B)
```

of the yet unknown function SET had previously been presented. And from this example, the template:

```
interaction([set, $ X, $ Y],Y]) :-
  is_instance(X,atom),
  is_instance(Y,list).
```

and the hypotheses:

```
hyp(set,required(number_of_arguments, set, 2), (1,[],[])).
hyp(set,required(type_of_arguments, set, [atom,list], (1,[],[])).
hyp(set,required(io_relation, set, second_argument), (1,[],[])).
```

had been generated. Since the hypotheses have only a confidence value of 1 (as indicated in Info) the confirmation strategy will be used first to test these hypotheses.

When exploring the LISP function SET with the parameter k being equal to 3, the following sequence of examples were generated by the simulation:

input	*expected result*	*correct result*
1. (SET 'C '(A B C))	*--> (A B C)*	*(A B C)*
2. (SET 'B '(F (F B))	*--> (F (F B))*	*(F (F B))*
3. (SET 'D '(E) 'A)	*--> (E)*	*ERROR*
4. (SET 'A)	*--> ERROR*	*ERROR*
5. (SET '(A B) '(A))	*--> (A)*	*ERROR*
6. (SET 'B 'A)	*--> A*	*A*

The examples 1. and 2. were generated using the confirmation strategy. They have the same number and the same types of arguments as the initial example from which the hypotheses were generated. The derived template could be applied, and therefore not much inferencing was required. Particular atoms and lists for the first and second arguments were randomly selected from a set of possible atoms and lists. The expected results were obtained by applying the template to the selected arguments. Since these results were correct, the confidence values of the hypotheses which were employed for generating these examples were incremented. After the second example, the hypotheses concerning the function SET thus had a confidence value of 3. The falsification strategy was then employed in order to test plausible alternative hypotheses.

The examples 3. and 4. were generated when testing the hypothesis about the required number of arguments. For the third example, a selection heuristic was used to generate the alternative hypothesis that the required number of arguments may be three. From this hypothesis an input was then constructed. For the newly introduced argument the same type was used as for the preceding argument. The expected result was derived from the hypothesis about the input output relation which says that the second argument will be returned. Since the actual result is an error message, the generated alternative hypothesis was rejected and added to the set of tested alternative hypotheses.

The fourth input tested the alternative hypothesis that the function SET can have one argument. An error message was expected since the hypothesis about the input output relation could not be applied without a second argument. Again the alternative hypothesis was rejected and added to the tested alternative hypotheses. After having tried 'one more' and 'one less' arguments all plausible alternative hypotheses concerning the number of arguments were tested.

The type hierarchy of the mental model was used for deriving alternative hypotheses for the types of arguments. In input 5., a list (instead of an atom) was used for the first argument. Since an error message was returned, the hypothesis that the first argument may be a list was rejected. In input 6., the type of the second argument was varied. Since the input was found to be correct, the hypothesis concerning the type of the second arguments was generalized. As the type hierarchy in the mental model did not offer any further alternatives, the hypothesis concerning the types of arguments was considered sufficiently tested. Exploration stopped, because no insufficiently tested hypotheses were left.

After the exploration the initial hypotheses about the function SET had been modified as follows:

```
hyp(required(number_of_arguments, set, 2),
   (7,[2],[1,3]).
hyp(required(type_of_arguments, set, [atom, s_expr]),
   (7,[[atom,atom],[atom,list]],
     [[list,atom],[list,list]]) ).
hyp(required(io_relation, set, second_argument),
   (7,[],[]) ).
```

Generating hypotheses in learning by exploration

If a learner is just given the name of some previously unknown function (e.g. the function SET), hypotheses have to be generated for the number and the types of arguments. For this purpose an already known function is selected. The respective specifications from this function are used as hypotheses for the new function. With these hypotheses an input to the LISP system is generated. If the input is found to be incorrect, different hypotheses are tried until a positive example is obtained. From a positive example a template is then constructed and the generated hypotheses can be further tested.

Combination of the different learning methods

Since sentences and examples are the units in the learning process and since the mental model is immediately updated, arbitrary sequences of sentences, examples and exploration episodes can be modeled. Obviously, a different sequence of instruction materials may yield different processing which may result in a different mental model.

If examples are studied after text (all the relevant clauses have been told) these clauses can be employed for constructing templates from the presented examples. Through the previously derived inferences, a considerably shorter explanation is obtained. Studying and inferencing from text thus facilitates the explication of knowledge from subsequent examples.

Quite different processing requirements arise for a text which is processed after examples have already been studied. In this case, hypotheses may have already been generated from the examples. In addition, templates may have been constructed. It is therefore possible that the text will contradict the already generated hypotheses. These incorrect hypotheses must consequently be deleted from the mental model (or appropriately modified). In order to maintain a consistent mental model, all inferences derived with incorrect hypotheses must also be deleted. On the other hand, if the text confirms the generated hypotheses, they can be asserted to be of type known.

The mental model which is being constructed when a user learns about or interacts with a system thus not only depends upon the learning episodes but also on the sequence of learning episodes. Thus, it comes as no surprise that almost every user may have a different mental model about a system. With the present simulation we can consequently investigate, how a user with some given mental model will tackle different tasks when interacting with the system.

Knowledge utilization

First we will briefly describe, how tasks are solved which more or less directly probe the correctness and completeness of the mental model. Such tasks are called verification tasks. Thereafter it will be described how tasks which are usually assumed to require how-to-do-it knowledge can be solved with the mental model. We will call these tasks application tasks.

Verification tasks

In a verification task either a sentence (sentence verification) or an example (example verification) is presented, and the student has to decide whether the presented item is correct or not. Verification tasks provide an ideal means for diagnosing the acquired mental model.

According to the simulation, verification tasks are solved by performing a consistency check. If the presented item is implied by the acquired knowledge (i.e. it is redundant in a logical sense), it will be judged as correct. If it is found that it contradicts the mental model, it will be judged as incorrect. If the presented item is neither redundant nor contradicting, it will be judged as unknown.

Application tasks

Previous research has shown that the knowledge for performing an application task can be modeled by goal driven production rules (e.g. Kieras and Polson, 1986). Such application tasks are defined through the goal which must be achieved with the particular computer system (e.g. text editor). In text editing the deletion of a word in an electronically stored text would be such a goal. This task is structurally similar to the deletion of an element in some list. In the following, we will discuss, how the goal of deleting an element from a list can be achieved by using the mental model of the LISP system. It will be shown how how-to-do-it knowledge can be formed thereafter.

For example, consider the programming task which requires a specific interaction to be performed with the LISP system: "Generate an input to the LISP system which deletes the atoms A and B from the list (A B C) which is bound to L, i.e. the function call should return the list (C)".

This task can be represented by:

```
solve_task((is_result([c]),
            is_argspec(l),
            bound_to(l,[a,b,c]) ),
           interaction(Input,Result)).
```

This goal of the task can be achieved by finding a specific interaction in the mental model which satisfies the three requirements: *is_result([c])*, *is_argspec(l)*, and *bound_to(l,[a,b,c])*. As previously pointed out, the mental model describes the structures and processes of the LISP system. Therefore, the programming task can be solved by finding a sequence of processes in the mental model. It is assumed that a mental model can be run forward and backward. Therefore, a sequence of processes can be mentally applied to *Result* in reverse order. When an Input is found which satisfies the above restrictions, the programming task is solved. Such a solution can be found by a means-end strategy. The more a mental model has been explicated, the less search is required for finding a solution. The following solution will be found by the simulation:

```
solve_task((is_result([c]),
           is_argspec(l),
           bound_to(l,[a,b,c]) ),
          interaction([rest, [rest,l]],[c]).

interaction([rest, [rest,l]],[c]) since
  funcall([rest, [rest,l]], rest, [[rest,l]]),
  eval_argspecs([[rest,l]],[[b,c]]),
  correct_function(rest,[[b,c]],[c]).
  eval_argspecs([[rest,l]],[[b,c]]) since
    eval([rest,l],[b,c]).
    eval([rest,l],[b,c]) since
      funcall([rest,l], rest, [l]),
      eval_argspecs([l],[[a,b,c]]),
      correct_function(rest,[[a,b,c]],[b,c]).
      eval_argspecs([l],[[a,b,c]]) since
      eval(l,[a,b,c]).
      eval(l,[a,b,c]) since
        bound_to(l,[a,b,c]).
```

Only a few inferences were needed for deriving this solution, since the previously explicated knowledge in the mental model was utilized. From this solution the following production rule is derived:

```
if solve_task(
  is_result(R),
  is_argspec(X),
  bound_to(X,[A,B|R])),
then is_input([rest,[rest,X]]).
```

This production rule states that if a task has to be solved in which a result R is to be obtained from a list of the form *[A,B|R]* which is bound to X then type: *[rest,[rest,X]]*. With this production rule as how-to-do-it knowledge similar tasks can be solved more efficiently. Production rules can also formed with knowledge of type hyp, i.e. with hypotheses. They are generated by the same mechanism which also generated the templates. Whereas templates can be seen as operational specializations for verification tasks, production rules are formed as operational specialization for application tasks. We have thus shown how how-to-do-it knowledge can be derived from the mental model, when specific tasks have to be solved.

Discussion

In previous research (Norman, 1983), it has been argued that the users' mental model (i.e. how-it-works knowledge) is very important for understanding their interactions with a system. Some experimental studies have shown the advantages of explicitly instructing a mental model (Kieras and Bovair, 1984). Also, it has been demonstrated that how-it-works-instructions can be more effective than how-to-do-it instructions (Schmalhofer, 1987), which may be due to how-it-works-instructions being more general (Catrambone, 1988).

Computer simulation models can be used to obtain a better understanding of these empirical results. However, previous models described only the knowledge which is sufficient to operate some system (Kieras and Polson, 1985) independent of any mental model. In a continuation of this line of research, we modeled the psychological processes of constructing a mental model when learning by being told, from examples, and by exploration. The simulation shows, how knowledge from related domains, hypotheses, and heuristic strategies are used for forming a mental model. Knowledge explication is seen as an important process for making the mental model more useful. The mental model which was learned for the domain of LISP is basically a cognitive LISP interpreter. The knowledge explication yields a more efficient cognitive LISP interpreter which has more structures and more efficient processes. When the mental model is applied to solve some task with the system, how-to-do-it knowledge is constructed. The proposed model thus presents a more complete picture of the cognitive complexities in learning to use a computer system.

Acknowledgements

This research was supported by grant Schm 648/1 from DFG. The cognitive model was initially programmed at the Institute for Cognitive Science of the University of Colorado, Boulder whose hospitality is greatly appreciated.

References

Card, S.K., Moran, T.P. and Newell, A. (1983). *The Psychology of Human-Computer Interaction.* Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Catrambone, R. (1988). *Specific versus General Procedures in Instructions.* Doctoral Dissertation, Psychology, University of Michigan.

Halasz, F.G. and Moran, T.P. (1983). Mental models and problem solving in using a calculator. In *Proceedings of CHI'83 Human Factors in Computing Systems.* New York: ACM.

Kieras, D.E. and Bovair, S. (1984). The role of mental model in learning to operate a device. *Cognitive Science,* 8, pp. 191-219.

Kieras, D. and Polson, P. (1985). An approach to the formal analysis of user complexity. *International Journal of Man-Machine Studies,* 22, pp. 365-394.

Mitchell, T. M., Keller, R. and Kedar-Cabelli, S. (1986). Explanation-based generalization: A unifying view. *Machine Learning,* 1, pp. 47-80.

Norman, D.A. (1983). Some observations on mental models. In D. Gentner and A.L. Stevens (Eds.), *Mental Models.* Hillsdale, NJ: Erlbaum.

Polson, P.G., Muncher, E. and Engelbeck, G. (1986). A test of a common elements theory of transfer. *CHI'86 Proceedings,* pp. 78-83.

Schmalhofer, F. (1987). Mental model and procedural elements approaches as guidelines for designing word processing instructions. In H. Bullinger and B. Shakel (Eds.), *Human-Computer Interaction INTERACT'87*. Amsterdam: North-Holland.

Schmalhofer, F. (1986). The construction of programming knowledge from system explorations and explanatory text: a cognitive model. In C.R. Rollinger and W. Horn (Eds.), *GWAI-86 and 2nd Austrian Artificial Intelligence Conference*. Heidelberg: Springer.

Ziegler, J.E., Hoppe, H.U. and Fähnrich, K.P. (1986). Learning and transfer for text and graphics editing with a direct manipulation interface. *Proceedings of CHI'86, Human Factors in Computing Systems,* Boston, pp. 72-77.