Interactive Support of Planning in a Functional, Visual Programming Language

C. MÖBUS, H-J. THOLE, O. SCHRÖDER
Department of Computational Science
Oldenburg University, P.O.Box 2503, 26111 Oldenburg, Germany
E-Mail: Claus.Moebus@arbi.informatik.uni-oldenburg.de

Abstract: Based on a theoretical framework of problem solving and knowledge acquisition, criteria for intelligent knowledge communication systems and help design are described. The ABSYNT Problem Solving Monitor for the acquisition of basic functional programming concepts in a visual language is designed according to these criteria. It incorporates *hypotheses testing* of solution proposals, and a learner model is designed to supply *user-adapted help*.

New is a third feature, which is presented in this paper: *Planning* programs with *goal nodes*. The learner can develop solution plans by postponing their implementation, and it is possible to test hypotheses with partial plans and with "mixed trees", existing of operator and goal nodes. The planning component of ABSYNT rests on a sound transformation approach (Bauer et al., 1987) that enables the derivation of functional programs from specifications. We hope to make derivational programming accessible even to beginners in very early stages of expertise.

Introduction

Intelligent knowledge communication systems supply the user with information which is sensitive to the actual problem solving situation and to the actual knowledge and intentions of the user. Developing such systems requires a variety of design problems, like when to supply remedial information, what to supply (what determines "good" help?), how to present the help information, an so on. The *acceptance* and *effectiveness* of knowledge communication systems critically depends on satisfactory solutions to these problems.

Two necessary requirements for developing an intelligent help system are:
• A theoretical framework based on detailed hypotheses about problem solving and learning. It is needed to support the design decisions for the system.
• A model of the learner´s actual knowledge state and its changes during the knowledge acquisition processes that occur while working with the system.

We work on a theoretical framework, which we call ISP-DL Theory (impasse - success - problem solving - driven learning theory (Möbus, Schröder & Thole, 1991; 1992; Möbus, Pitschke & Schröder, 1992). It is intended to describe continuous problem solving and knowledge acquisition processes of learners by integrating the theoretical concepts of impasse-driven learning (Laird, Rosenbloom & Newell, 1986; 1987; Newell, 1990; Rosenbloom et al., 1991; van Lehn, 1988; 1989; 1990; 1991), success-driven learning (e.g., Anderson, 1983; 1986; 1989; Wolff, 1987; 1991), and different problem solving phases or stages (according to Gollwitzer, 1990; 1991). Briefly, the ISP-DL Theory states that a problem solving process may be structured into the following phases: The problem solver (PS) *deliberates* with the result of choosing a goal to persue; then a *plan* to reach the goal is created, the plan is *executed*, and finally the obtained result is *evaluated*. *Impasses* might result at several points in this process: The PS might not be able to choose a goal, or the plan cannot be created, or execution is not possible, or the obtained result is not satisfying. The PS reacts to an impasse by problem solving, using weak *heuristics*: looking for help, asking, cheating, and so on. As a result, the PS may overcome the impasse and acquire new knowledge (impasse-driven learning). But alternatively, the information obtained may not be helpful but confusing, complicating things, and so on. So instead of resolving the impasse, the learner might encounter a secondary impasse (Brown & van Lehn 1980). Finally, if a problem has been successfully solved without impasses, then the knowledge used is optimized (success-driven learning).

Thus according to ISP-DL Theory there is a threefold synchronization problem for help information:

• According to the theory, the learner will look for and appreciate help if she or he is caught in an impasse. So help should be given at *impasse time* (that is, *offered* to the learner *without interrupting him*).

• In order to be helpful at impasses, help should acknowledge the actual *knowledge state* of the learner. Help should be *user-oriented*.

• Help should be provided at different phases of problem solving (*deliberating*, *planning*, *executing*, and *evaluating*) because impasses may arise at all phases. Help should be *problem phase oriented*.

Centered around the ISP-DL Theory and models of knowledge acquisition, we developed two help systems: The ABSYNT Problem Solving Monitor (PSM) supports functional programming in a visual language (Möbus, Schröder & Thole, 1991; 1992), and PETRI-HELP supports modelling concurrent or distributed processes with condition-event Petri nets (Möbus, Pitschke & Schröder, 1992). So we try to realize the ISP-DL Theory and its implications for help system design in two different domains. Our work related to the first and the second help synchronization problem has been described elsewhere (Möbus, Schröder & Thole, 1992).

This paper focuses on the incorporation of the problem solving phases of *deliberating* and *planning* into ABSYNT. We will demonstrate and discuss an approach to supply interactive support and help for the user´s processes of deliberating and planning while constructing functional programs in ABSYNT.

The ABSYNT Problem Solving Monitor

ABSYNT ("Abstract Syntax Trees") is a functional, visual programming language based on ideas stated in an introductory computer science textbook (Bauer & Goos, 1982). ABSYNT is a tree representation of pure LISP and is aimed at supporting the acquisition of basic functional programming skills, including abstraction and recursive systems. The motivation and analysis of ABSYNT with respect to properties of visual languages and cognitive science principles is described in (Möbus & Schröder, 1990).

The ABSYNT PSM provides an *iconic programming environment*. Its main components are a visual *editor*, a visual *trace*, and a *help component*: a *hypotheses testing environment*. The design of the ABSYNT PSM is motivated by the ISP-DL Theory in several respects:

• As recommended by the ISP-DL Theory, the ABSYNT PSM does not interrupt the PS, but *offers* help for the PS to overcome impasses.

• According to the ISP-DL Theory, the PS should be able to make use of his pre-knowledge at impasses as much as possible. In the ABSYNT PSM, this principle is realized by the *hypotheses testing approach*. The learner may state hypotheses about which part of his current solution proposal he considers correct. The system then analyzes the hypothesis and gives feedback. The student can also ask the system for completion proposals (see below). Another reason for the hypotheses testing approach is that in programs it is usually not possible to absolutely localize bugs. Often the bug consists of an inconsistency between program parts, and there are several ways to fix it. The hypotheses testing approach leaves the decision how to change a buggy program to the *PS*.

• According to the ISP-DL Theory, help should be aimed at the actual phase of problem solving. The ABSYNT PSM supports the problem solving phases of *planning*, *executing*, and *evaluating* solution proposals. A solution proposal may be *planned* first by using goal nodes. So the learner may create a plan and test hypotheses about it without bothering about its implementation at this stage. The *implementation* of the goals (thus creating an executable program) may be done later. *Evaluation* is again supported by hypothesis testing.

Figure 1 depicts snapshots from the ABSYNT PSM. Figure 1a shows the *visual editor* where ABSYNT programs can be created. There is a head window and a body window. On the left side of Figure 1a, there is the tool bar of the editor: Nodes are connected with the line. The bucket is for deleting nodes and links. The hand is for moving, and the pen for naming nodes. Next, there is a constant, parameter and "higher" operator node (to be named by the learner, using the pen tool). Constant and parameter nodes are the *leaves* of ABSYNT trees. The "goal" node will be explained below. Then the primitive operator nodes follow. Editing is done by selecting nodes with the mouse and placing them in the windows, and by linking, moving, naming, or deleting them. Nodes and links can be created *independently*: If a link is created before the to-be-linked nodes are edited, then shadows are automatically created at the link ends. They serve as place holders for nodes to be edited later.

Constant, parameter and operator nodes are *implementation* nodes. A syntactically correct ABSYNT program is runnable if it consists only of implementation nodes. Implementation nodes have three horizontal parts: an input stripe, a name stripe, and an output stripe. (Constant nodes have only two stripes because name and output are identical.) In the *visual trace* of the ABSYNT PSM (not depicted), input and output stripes are filled with computation goals and obtained values, so each computational step of the ABSYNT interpreter can be visualized (Möbus & Schröder, 1990).

Making help adaptive to the actual phase of problem solving

As already indicated, in ABSYNT there are also *goal* nodes designed to support the hypothetical problem solving phases of *deliberating* and *planning*. Clicking on the "goal" symbol in the tool bar (Figure 1a, on the left) causes the tool bar to switch to the actual goal nodes. In general, goals are facts the problem solver wants to become true. Here, goal nodes represent abstract plan fragments which are to be implemented later in several possible ways by implementation nodes or subtrees. Goal nodes have a different shape and no iconic internal structure. In Figure 1a, "LIST EMPTY" and "CASE" are examples of goal nodes. Each goal node is precisely defined as a predicative description for the yet to be implemented program fragments. The "LIST EMPTY" node represents the goal to test if a list is empty (Formally: goal LIST EMPTY (list l) bool: that bool x : x = (l = nil). So LIST EMPTY represents the goal to determine for a list l that boolean value which results from evaluating "l = nil"). The "CASE" node represents the goal to program conditionalized expressions, that is, condition-expression pairs (Formally: goal CASE (bool p_1, value a_1, bool p_2, value a_2,..., bool p_n, value a_n) value: if p_1 then a_1 else if p_2 then a_2 else ... if p_n then a_n fi ... fi fi).

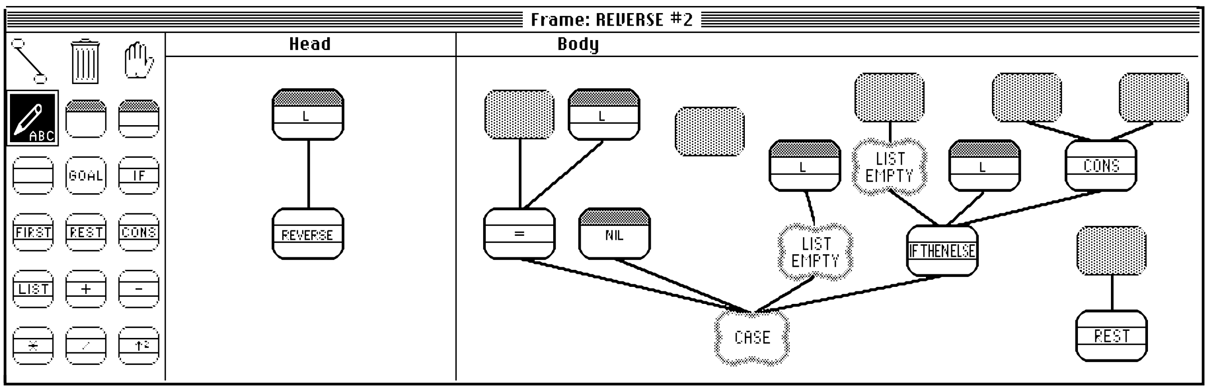


Figure 1a. Student´s erroneous and overly complicated proposal to the "reverse" problem in the visual editor

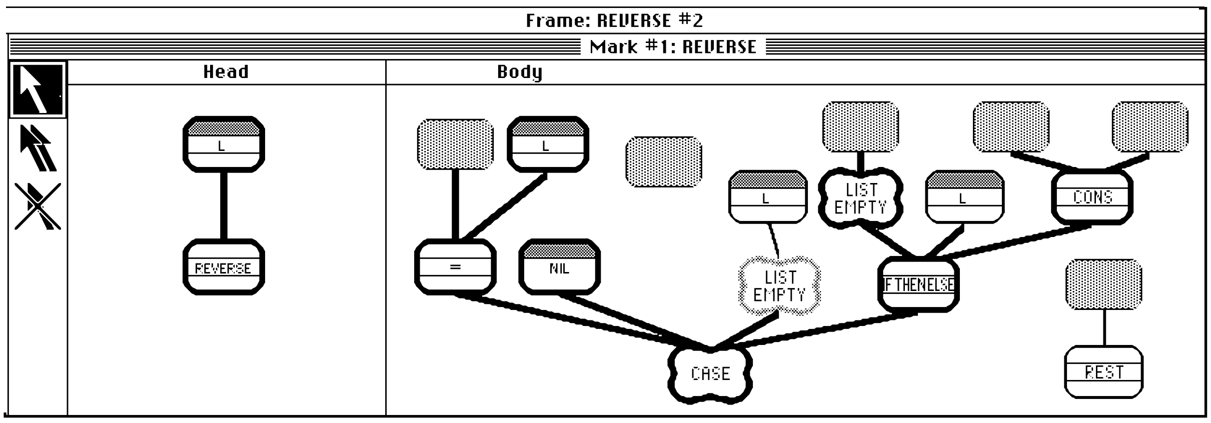


Figure 1b. Student´s hypothesis (bold nodes and links)

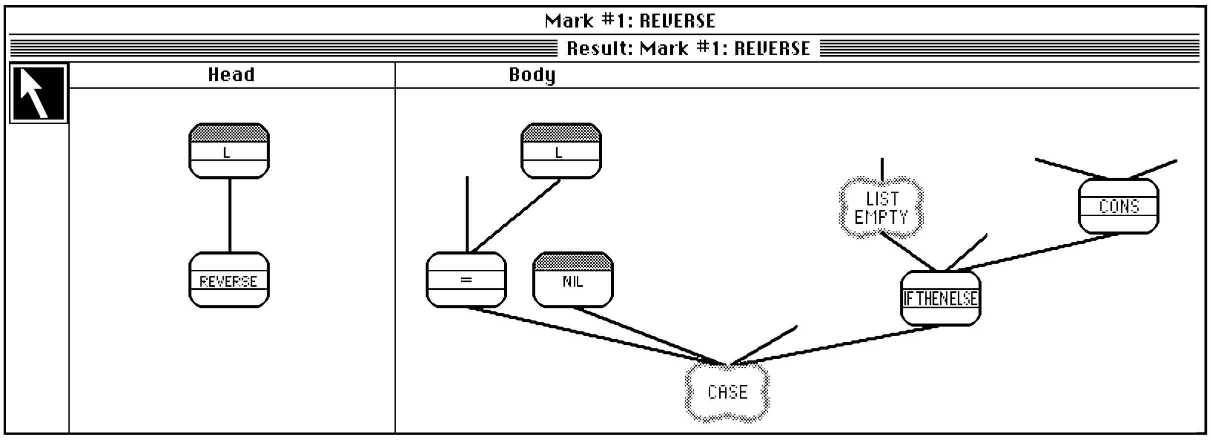


Figure 1c. Positive Feedback of the ABSYNT system to student´s hypothesis

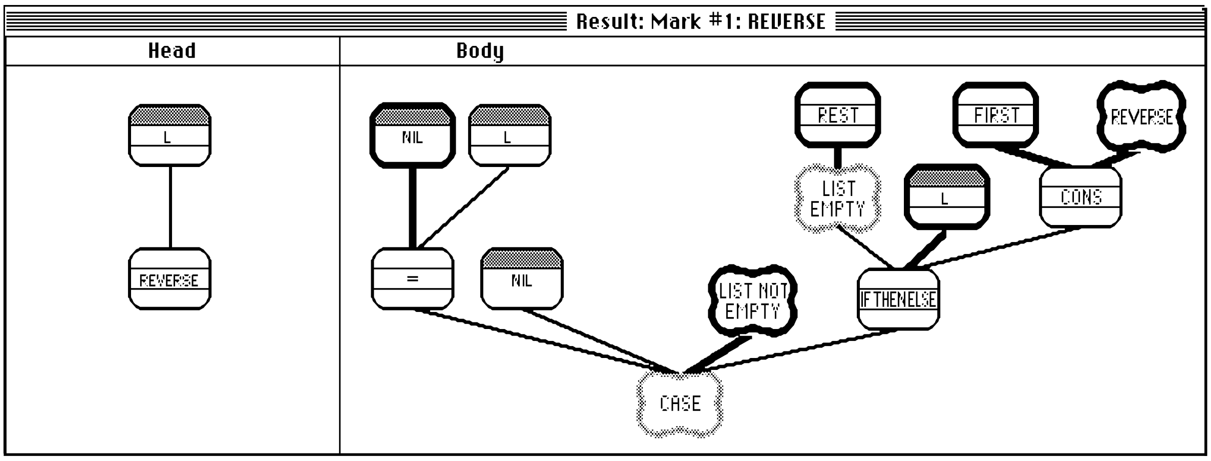


Figure 1d. Completion proposals of the ABSYNT system on student demand

The ABSYNT goal nodes are based on a task analysis which applies the *transformation* approach developed in the Munich CIP Project (Bauer et al., 1987; Partsch, 1990). The transformation approach ensures that a solution can be derived to a given task that is correct with respect to the task description. Currently ABSYNT supports 42 programming tasks. For each task, there is a top level goal node and a collection of lower goal nodes with predicative and verbal descriptions. Data types are numbers, truth values, and lists.

In Figure 1a, a wrong solution proposal for an ABSYNT program reversing a list is just being created. There are nodes not yet linked or even completely unspecified (shaded areas). As Figure 1a shows, goal nodes and implementation nodes can be mixed ("mixed trees") within a proposal. The solution proposal in Figure 1a means:

```
If L is equal to the value of a yet unknown expression,
then the value of REVERSE is NIL,
else    if L is an empty list,
        then    if the value of a yet unknown expression is the empty list,
                then the value of REVERSE is the value of L,
                else the value of REVERSE is obtained by CONSing the values of two yet
                unknown expressions together.
```

In the *hypotheses testing environment* the learner may state hypotheses (by marking parts of the program so they get bold) about the correctness of a solution proposal or parts thereof for a given programming task. The hypothesis is: "It is possible to embed the boldly marked fragment of the program in a correct solution to the current task!". In Figure 1b the learner stated a hypothesis which covers a fragment of the proposal created so far for the "reverse" programming task. The hypothesis contains goal nodes and implementation nodes. The system analyzes the hypothesis and recognizes it as embeddable, indicating this by returning a copy of the hypothesis to the student (Figure 1c). If this information is not sufficient for solving the impasse, the student may ask the system for completion proposals at the open links. In Figure 1d, the student asked for and received six completions (bold). Two of them are goal nodes, the others are implementation nodes. The "REVERSE" goal node represents the task goal. As far as possible, the system tries to generate completions consistent with the student´s proposal. At one point, the system disagrees with the student´s proposal: The system proposes "LIST NOT EMPTY" at the third input link of the CASE node, whereas the student´s original proposal contains "LIST EMPTY" at this point (Figure 1a). Internally, the system has created a complete solution but the student always gets only minimal information. If the learner states a hypothesis that cannot be embedded in a correct solution (not shown here), then the learner receives the message that the hypothesis cannot be completed to a solution known by the system.

The hypotheses testing environment allows to test and get completions for plan and program fragments. It is the most significant aspect where the ABSYNT PSM differs from other systems designed to support the acquisition of functional programming knowledge, like the LISP tutor (Anderson & Swarecki, 1986; Anderson, Conrad & Corbett, 1989; Corbett & Anderson, 1992), the SCENT advisor (Greer, 1992; Greer, McCalla & Mark, 1989), and the ELM system (Weber, 1989; 1992). This is true also for the difference of ABSYNT and the visual data flow programming system "Function Machines" (Feuerzeig, Richards & Roberts, 1989). As indicated, one reason for the hypotheses testing approach is that in programming a bug usually cannot be absolutely localized. Hypotheses testing leaves the decision which parts of a buggy solution proposal to keep to the student and thereby provides a rich data source about the learner´s knowledge and intentions. Single subject sessions with the ABSYNT PSM revealed that hypotheses testing was heavily used. It was almost the only means of debugging, despite the fact that the subjects had also the visual trace available. This is partly due to the fact that in contrast to the trace, hypotheses testing does not require a complete ABSYNT program solution. Hypotheses testing is possible with incomplete solutions, with goal nodes, and with mixed terms. So the student may obtain feedback whether he is on the right track at very early planning stages.

The answers to the learner´s hypotheses are generated by rules defining a *goals-means-relation (GMR)* (Levi & Sirovich, 1976; Nilsson, 1980). A subset of these rules may be viewed as "pure" expert domain knowledge not influenced by learning. Thus we call this set of rules EXPERT. These rules are able to analyze and to synthesize several millions of plans and solutions for the 42 tasks (Möbus, 1991; Möbus & Thole, 1990). We think that such a large solution space is necessary because we observed that especially novices often construct unusual solutions due to local repairs. Currently, EXPERT contains about 1300 planning rules and implementation rules. The planning rules elaborate goals, and the implementation rules describe how to realize goals by ABSYNT implementation nodes. The goal decomposition done by the planning rules follows the CIP transformation approach mentioned earlier. The formal description of a task or a programming goal consists of a specification, which is transformed stepwise into a solution plan. Figure 2 illustrates one transformation step to obtain a solution plan for the task "add by add 1: Add two natural numbers using only addition and subtraction by one." The ABSYNT tree corresponding to the result of this transformation step is also shown in Figure 2. The dots in Figure 2 indicate several necessary transformation steps such as case introduction, predicate introduction, and folding. The Δ-symbol means sequential conjunction.

Making help adaptive to the problem solver´s actual knowledge

The completions shown in Figure 1d (bold program fragments) were generated by EXPERT rules. EXPERT analyzes and synthesizes solution proposals but is not *adaptive* to the learner´s knowledge. Usually EXPERT is able to generate a large set of *possible* completions. For example, EXPERT could generate a large number of alternatives for the "LIST NOT EMPTY" goal node in Figure 1d. Thus the problem is to *select* the most appropriate completion proposal. This is the function of a model of the learner´s actual knowledge state in ABSYNT.

We developed such a model which we call a *State Model* since it represents the successive knowledge states of a PS as he moves from a novice to an expert in the ABSYNT domain. It consists of rules derived from EXPERT. The State Model should offer a completion proposal to the PS which is maximally *consistent* with the learner´s current knowledge state. This should minimize the learner´s surprise to feedback and completion proposals. The State Model is designed as an integrated part of the ABSYNT PSM. It represents the actual hypothetical domain knowledge of the learner at different points in the knowledge acquisition process. The hypothetical domain knowledge is organized as a partial order of *micro rules*, *schemas*, and specific *cases*. Micro rules represent knowledge newly acquired by *impasse-driven learning* but not yet optimized. They describe small planning or implementation steps in the ABSYNT domain. Schemas and cases are created by rule composition according to the resolution method. The State Model is created and updated by automatically inspecting the single editing steps performed by the user while constructing ABSYNT programs.

Detailed descriptions of the State Model are provided in (Möbus, Schröder & Thole, 1991; 1992).

task: add by add 1 "add two natural numbers using only add1 und sub1"
task specification: **that nat** x: x = a + b

●
case introduction
●

conditional inference under constraint b > 0 :

expression: **funct** addadd_1 **(nat** a,b) **nat:**
if b = 0 **then** a
if b > 0 **then that nat** x:
exists nat x': [a + (b - 1) = x'] Δ [x' + 1 = x]

*applying the **CIP-Rule "choice and quantification"***

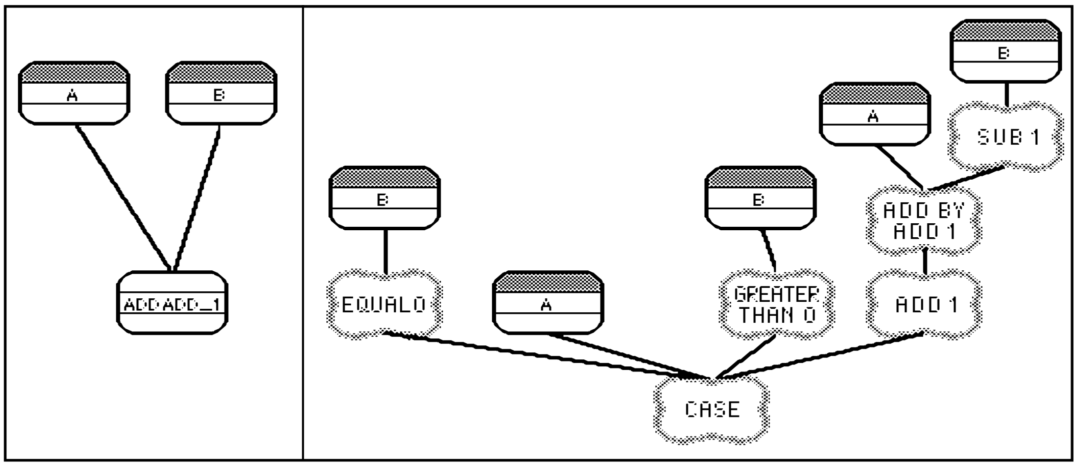
gives the expression: **funct** addadd_1 **(nat** a,b**) nat:**
if b = 0 **then** a
if b > 0
then add1 **(that nat** x': x'=a + sub1(b) **)**
specification of subtask

sub1(b) or b - 1

CIP-Rule "choice and quantification":

```
that n x: exists  m y:(P(y) Δ f(y) = x)
 Δ
 ||          | EQUIV(m,=)
 ||__________| "=" is
 ||          | equal predicate
 ||          | of sort m
 V
f(that m y: P(y))
with bindings:  rule  / derivation
   x  / x
   y  / x'
   P(y) / a + (b - 1) = x'
   f(y) / x' + 1         or  add1(x')
```

This expression corresponds to the following ABSYNT head and body tree:



●
folding
●

Figure 2. One transformation step on the way to a solution plan for the task "add by add 1 "

Conclusions

The ISP-DL Theory is a theoretical framework of problem solving and knowledge modification which has important implications for the design and development of knowledge communication systems. Specifically, according to the theory there are three requirements for information if it is intended to be helpful: Information will only be appreciated if received at impasse time, information has to be aimed at the current phase of problem solving, and it must be consistent with the actual knowledge state of the PS. We described our realizations of these requirements within the ABSYNT Problem Solving Monitor designed to support the acquisition of functional programming skills. In ABSYNT, the PS may state hypotheses and get completion proposals from the system on demand (= *help at impasse time*). The PS may plan with goal nodes, implement the plan afterwards, and get goal node completions and implementation node completions as well (= *help at different problem solving phases*). Furthermore, completion proposals are designed to be adaptive to the actual learner´s knowledge by being controlled by a model of the actual learner´s knowledge state (= *knowledge state adapted help*).

In this paper we focussed on planning with ABSYNT which is based on the transformational approach of the Munich CIP Project. Incorporating planning into ABSYNT has benefits from three perspectives:

- From the learner´s point of view, the benefit of planning with goal nodes is that hypotheses testing is possible already in the planning phase, and at *very early stages* of solution development in general. So the learner will get information whether she or he is "on the right track" before starting with the implementation. For example, the learner may make use of the recursion concept by creating a recursive plan without bothering about its runnable implementation at this stage.
- In a preliminary empirical investigation a single subject worked through the sequence of ABSYNT tasks, using the goal nodes. Our observations are that she made much use of the goal nodes, judged them as convenient in many cases, and did much hypotheses testing based on partial plans and (sometimes) on mixed trees. More empirical work concerning the acceptability and usefulness of goal nodes is under progress.
- From a help system design point of view, the benefit of integrating goal nodes into the ABSYNT PSM is that in addition to hypotheses testing with goal nodes and receiving goal completions, it will be possible to offer *planning rules* as help to the learner.
- Finally, from a psychological point of view, the benefit of planning with goal nodes is that objective *data about the planning process* can be obtained in addition to verbalizations. Usually such data about planning and reasoning "before" the implementation are not available. We work on the augmentation of our *State Model* so it will be able to account for the use of goal nodes, so *user-oriented* completion proposals will be possible on the planning level.

References

Anderson, J.R. (1983). *The architecture of cognition*. Cambridge: Harvard University Press.

Anderson, J.R. (1986). Knowledge compilation: The general learning mechanism. In R.S. Michalski, J.G. Carbonell & T.M. Mitchell (eds). *Machine Learning, Vol. II.* Los Altos: Kaufman, 289-310.

Anderson, J.R. (1989). A theory of the origins of human knowledge. *Artificial Intelligence, 40,* 313-351.

Anderson, J.R., Conrad, F.G. & Corbett, A.T. (1989). Skill acquisition and the LISP tutor. *Cognitive Science, 13,* 467-505.

Anderson, J.R. & Swarecki, E. (1986). The automated tutoring of introductory computer programming. *Communications of the ACM, 29,* 842-849.

Bauer, F.L., Ehler, H., Horsch, A., Möller, B., Partsch, H., Paukner, O. & Pepper, P. (1987). *The munich project CIP, Vol. II: The program transformation system CIP-S.* Berlin: Springer (LNCS 292).

Bauer, F.L. & Goos, G. (1982). *Informatik (Vol. 1)*. Berlin: Springer, (3rd ed.).

Brown, J.S. & van Lehn, K. (1980). Repair Theory: A generative theory of bugs in procedural skills. *Cognitive Science, 4,* 379-426.

Corbett, A.T. & Anderson, J.R. (1992). Student modeling and mastery learning in a computer-based programming tutor. In C. Frasson, G. Gauthier & G.I. McCalla (eds). *Intelligent Tutoring Systems. Proceedings ITS 92.* Berlin: Springer, (LNCS 608), 413-420.

Feuerzeig, W., Richards, J. & Roberts, N. (1989). *Function machines - user manual*. BBN Laboratories.

Gollwitzer, P.M. (1990). Action phases and mind-sets. In: E.T. Higgins & R.M. Sorrentino (eds). *Handbook of Motivation and Cognition: Foundations of Social Behavior, Vol.2,* 53-92.

Gollwitzer, P.M. (1991). *Abwägen und Planen*. Göttingen. Toronto: Verlag für Psychologie.

Greer, J. (1992). *Granularity and context in learning.* Invited talk at the 2nd international conference on intelligent tutoring system. Montreal, Canada.

Greer, J., McCalla, G.I. & Mark, M.A. (1989). Incorporating granularity-based recognition into SCENT. In *Proceedings 4th Int. Conference on Artificial Intelligence and Education*. Amsterdam: IOS-Verlag.

Laird, J.E., Rosenbloom, P.S. & Newell, A. (1986). *Universal subgoaling and chunking. The automatic generation and learning of goal hierarchies.* Boston: Kluwer.

Laird, J.E., Rosenbloom, P.S. & Newell, A. (1987). SOAR: An architecture for general intelligence. *Artificial Intelligence, 33*, 1-64.

Levi G. & Sirovich, F. (1976). Generalized and/or-graphs. *Artificial Intelligence, 7*, 243-259.

Möbus, C. (1991). The relevance of computational models of knowledge acquisition for the design of helps in the problem solving monitor ABSYNT. In R.Lewis & S.Otsuki (eds). *Advanced Research on Computers in Education*.. Tokyo, Japan, 18-20 July, 1990, Elsevier Science Publishers B.V. (North-Holland), 137-144.

Möbus, C., Pitschke, K. & Schröder, O. (1992). Towards the theory-guided design of help systems for programming and modelling tasks. In C. Frasson, G. Gauthier & G.I. McCalla (eds). *Intelligent Tutoring Systems. Proceedings ITS 92*. Berlin: Springer (LNCS 608), 294-301.

Möbus, C. & Schröder, O. (1990). Representing semantic knowledge with 2-dimensional rules in the domain of functional programming. In: P.Gorny & M. Tauber (eds). *Visualization in Human-Computer Interaction. 7th Interdisciplinary Workshop in Informatics and Psychology*. Schärding, Austria, May 1988, Berlin-Heidelberg-NewYork: Springer, (LNCS 439), 47-81.

Möbus, C., Schröder, O. & Thole, H.-J. (1991). Runtime modeling the novice-expert shift in programming skills on a rule-schema-case continuum. In: J. Kay & A. Quilici (eds). *Proceedings of the IJCAI Workshop W.4 Agent Modelling for Intelligent Interaction. 12th Int. Joint Conf. on Artificial Intelligence.* Darling Harbour, Sydney, Australia, 137-143.

Möbus, C., Schröder, O. & Thole, H.-J. (1992). A model of the acquisition and improvement of domain knowledge for functional programming. *Journal of Artificial Intelligence in Education 3(4)*, 477-493.

Möbus, C. & Thole, H.-J. (1990). Interactive support for planning visual programs in the problem solving monitor ABSYNT: Giving feedback to user hypotheses on the basis of a goals-means-relation. In: D.H. Norrie & H.-W. Six (eds). *Computer Assisted Learning. Proceedings of the 3rd International Conference on Computer-Assisted Learning ICCAL 90.*Hagen, F.R.Germany. Heidelberg: Springer, (LNCS 438), 36-49.

Newell, A. (1990). *Unified theories of cognition*. Cambridge: Harvard University Press.

Nilsson, N.J. (1980). *Principles of artificial intelligence.* Palo Alto: Tioga Publ. Co.

Partsch, H.A. (1990). *Specification and transformation of programs: A formal approach to software development.* Berlin: Springer.

Rosenbloom, P.S., Laird, J.E., Newell, A. & McCarl, R. (1991). A preliminary analysis of the SOAR architecture as a basis for general intelligence, *Artificial Intelligence, 47*, 289-305.

Van Lehn, K. (1988). Toward a theory of impasse-driven learning. In: H. Mandl & A. Lesgold (eds). *Learning Issues for Intelligent Tutoring Systems*. New York: Springer, 19-41.

Van Lehn, K. (1989). *Learning events in the acquisition of three skills. Proc. 11th conf. cognitive science society.* Ann Arbor, Michigan, 434-441.

Van Lehn, K. (1990). *Mind bugs: The origins of procedural misconceptions.* Cambridge: MIT Press.

Van Lehn, K. (1991). Rule acquisition events in the discovery of problem solving strategies. *Cognitive Science, 15*, 1-47.

Weber, G. (1989). Automatische kognitive Diagnose in einem Programmier-Tutor. In D. Metzing (ed). *Künstliche Intelligenz GWAI 89*. Berlin: Springer, 331-336.

Weber, G. (1992). Analogien in einem fallbasierten Lernmodell. In K. Reiss, M. Reiss & H. Spandl (Hrsg). *Maschinelles Lernen - Modellierung von Lernen mit Maschinen.* Berlin, Heidelberg: Springer, 143-175.

Wolff, J.G. (1987). Cognitive development as optimisation. In L. Bolc (ed). *Computational Models of Learning*. Berlin: Springer, 161-205.

Wolff, J.G. (1991). *Towards a theory of cognition and computing.* Chichester: Ellis Horwood.

Acknowledgement

We thank J. FOLCKERS for implementing the goal nodes in ABSYNT.