Diagnosis of Intentions and Interactive Support of Planning in a Functional, Visual Programming Language

Claus Möbus, Heinz-Jürgen Thole, Olaf Schröder *

Department of Computational Science
University of Oldenburg, 26111 Oldenburg, Germany
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*. According to our theory, the use of these nodes is an indicator of the planner's *intentions*. They have to be replaced later by runnable ABSYNT operators or program trees by the planner. Furthermore the learner can test hypotheses about the correctness of ABSYNT programs containing operator and goal nodes. The planning component of ABSYNT rests on a sound transformation approach [6] that enables the derivation of functional programs from specifications. The ABSYNT goal nodes are derived from corresponding transformation rules (see appendix). Though the transformation approach is technically sound it is not accessible to novices and sometimes even to experts. By offering goal nodes for hypotheses testing in the problem solving phases of deliberating and planning, we hope to make derivational programming accessible even to beginners at very early stages of expertise.

Keywords: intention diagnosis, derivational programming, hypotheses testing, support of planning and deliberation

Introduction

Intelligent knowledge communication systems, like help systems, tutoring systems, and problem solving monitors, are expected to 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 remedial information, an so on. The *acceptance* and *effectiveness* of knowledge communication systems critically depends on satisfactory solutions to these problems.

In order to tackle these problems, a system of hypotheses about learners processes of problem solving and knowledge acquisition is necessary. Such a theoretical framework may help to support design decisions for several components of an intelligent knowledge communication system. For example, it may determine what kind of help to supply, and when to supply it, given certain features of the problem solving situation and of the learner.

We work on such a theoretical framework, which we call ISP-DL Theory (impasse - success - problem solving - driven learning theory) [25, 26]. According to the theory, the stream of (internal and external) actions of a problem solver consists of different problem solving phases: *deliberating*, *planning*, *executing*, and *evaluating*. Impasses, which are possible at each phase,

* We thank Jörg Folckers for implementing the user interface of ABSYNT.

may lead to problem solving and to the subsequent acquisition of new knowledge. Successful problem solving leads to the optimization of the knowledge applied.

The ISP-DL Theory implies several design criteria for the development of an intelligent knowledge communication system. We develop two systems: The ABSYNT Problem Solving Monitor (PSM) supports functional programming in a visual language [22, 25, 26, 27]. PETRI-HELP supports modelling concurrent of distributed processes with condition-event Petri nets [23, 36]. So we try to realize the ISP-DL Theory and its implications for the design of a knowledge communication system in two different domains.

The aim of this paper is to show how the problem solving phases of *deliberating* and *planning* can be supported by a knowledge communication system. While earlier versions of the ABSYNT PSM addressed the problem solving phases of *executing* and *evaluating*, here we will demonstrate and discuss an approach to supply interactive support and help for the learner´s processes of *deliberating* and *planning* while constructing functional programs within the ABSYNT PSM. First ideas of this approach are presented in [28]. The paper has three parts: In the first part we will briefly describe the ISP-DL Theory and the criteria recommended by it for the design of a knowledge communication system. In the second part the ABSYNT Problem Solving Monitor and its relationship to the design criteria is described, with a focus on the realization of deliberating and planning in ABSYNT. In addition, our work in making help information adaptive to the learner´s knowlededge state is briefly described. Finally, some conclusions are given.

A Theoretical Framework of Problem Solving and Knowledge Acquisition

The ISP-DL Theory is intended to describe continuous problem solving and knowledge acquisition processes of a learner as it occurs in a sequence of, for example, programming sessions. It is an attempt to integrate the theoretical concepts of impasse-driven learning [18, 19, 33, 37, 38, 39], success-driven learning, e.g. [1, 2, 3, 29, 31, 41, 42], and different problem solving phases according to [14, 15]. The ISP-DL Theory has three components:

• *Problem solving phases.* 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 certain goal [30] to pursue; then a *plan* to reach the goal is created, the plan is *executed*, and finally the obtained result is *evaluated.*

• *Acquisition of new knowledge. Impasses* might result at several points in the problem solving process: The PS might not be able to choose a goal, or a plan cannot be created, or its 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 [8].

• *Improvement of existing knowledge.* If a problem has been successfully solved without impasses, then the knowledge applied is optimized (success-driven learning) so it can be used more effectively the next time. For example, the number of control decisions and subgoals to be set may be reduced.

The ISP-DL Theory leads to several design principles for a knowledge communication system:

• According to the theory, the learner will look for and appreciate help if she or he is caught in an *impasse*. Without an impasse there is no need for help. So the system should not interrupt the learner, but *offer* help on request.

• According to the theory, the learner should be prevented from trapping into secondary impasses which may lead away from the original problem solving. This may be done by letting the learner make use of his *pre-knowledge* at impasses as much as possible. This principle may be realized in two ways:

> • To let the learner *test hypotheses* about her or his solution proposals. This means that the *learner* may decide which *part* of a proposal she or he considers correct. The learner can ask the system for analysis of the hypothesis and for completion proposals. This leaves the activity on the learner's side, and the learner is not disturbed by unwanted system interventions and comments.
>
> • To adapt remedial information and help to the actual knowledge state of the learner. Help should be *knowledge state oriented*, requiring a learner model.

• Finally, according to the theory, information useable as help should be provided for the different phases of problem solving because impasses may arise at all phases. So a help system should support *deliberating*, *planning*, *executing*, and *evaluating* solution proposals. Help should be *problem phase oriented.*

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 [7]. 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 design of ABSYNT as a visual programming language was based on

> • *two alternative runnable specifications* of the ABSYNT interpreter [24] which were developed according to cognitive science principles and constraints [20],
>
> • empirical studies concerning the mental representation of and misconceptions about functional programs.

This work served to prepare the development of the ABSYNT PSM according to principles of visual learning environments [13]. ABSYNT is analyzed with respect to properties of visual languages in [24].

The ABSYNT PSM provides an *iconic programming environment* [9]. 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 the following ways:

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

• 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 hypothesize which part of his current solution proposal he considers correct. The system then analyzes the hypothesis and gives feedback. The PS can also ask the system for completion proposals (see below). A second 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 provided at different phases of problem solving. The ABSYNT PSM enables and supports all problem solving phases at least to some extent: *deliberating* with the result of choosing a programming task to do, *planning* a solution to it, *executing* the plan, and *evaluating* the solution proposal. In the ABSYNT PSM, the *deliberation* phase corresponds to choosing a programming task. It is supported by the system's ability to propose subtasks. The *planning* phase corresponds to creating a solution proposal by using *goal nodes* (see below). So the learner may create a plan and test hypotheses about it without bothering about its implementation at this point. The *implementation* of the goals (thus creating an executable program) may be done later, using *implementation nodes*. Both planning and executing are supported because the learner may receive goal nodes or implementation nodes as completion proposals from the system on request. Finally, *evaluation* corresponds to hypotheses testing and to using the visual trace.

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: The line is for connecting nodes. The bucket is for deleting nodes and links. The hand is for moving nodes, the pen for naming nodes, and the question mark for getting descriptions of them. The "goal" tool will be explained below. 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. Then several primitive operator nodes follow ("if", "first", "rest", "cons", "list", "+", "-", "*", ...). 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. Shadows may also be created by clicking into a free region of a window.
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 [24].

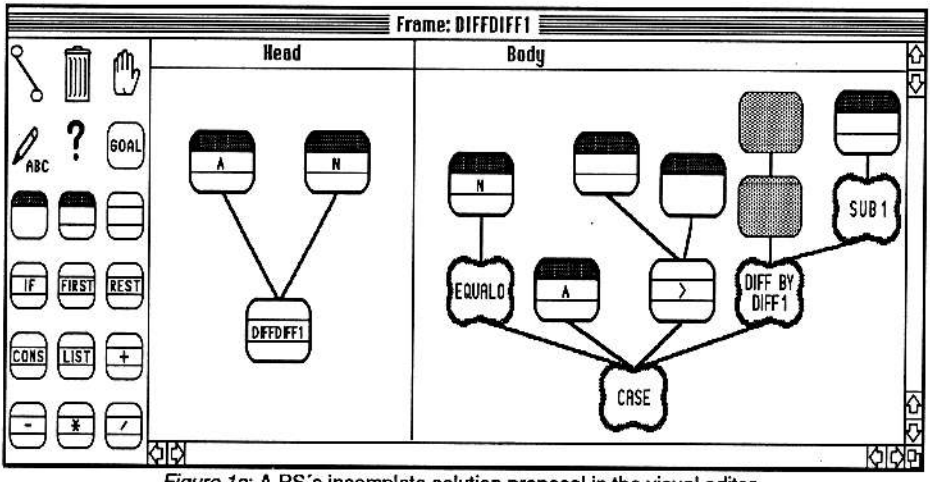


Figure 1a: A PS's incomplete solution proposal in the visual editor

Figure 1: Snapshots of problem solving with ABSYNT: A PS's incomplete proposal to the "diff by diff 1" problem (Figure 1a), the PS's hypothesis proposal (Figure 1b), feedback (Figure 1c) and completion proposal of the ABSYNT system (Figure 1d). *Continued on the next pages*

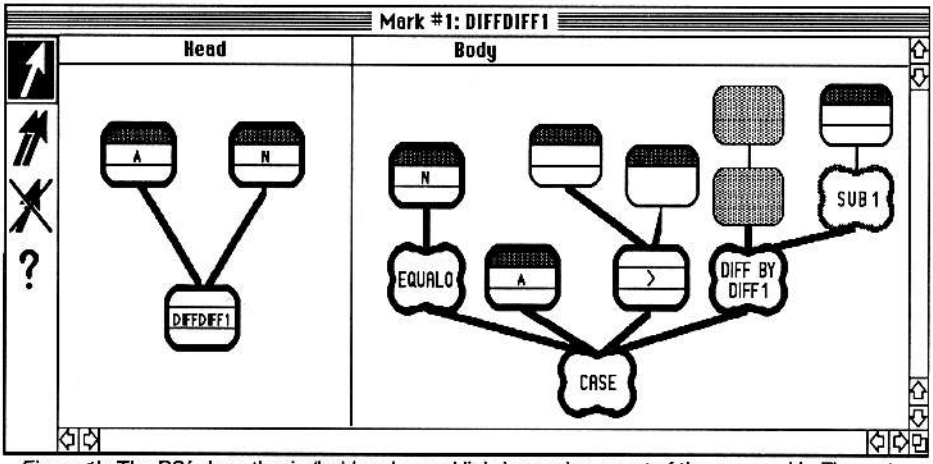


Figure 1b: The PS's hypothesis (bold nodes and links) covering a part of the proposal in Figure 1a

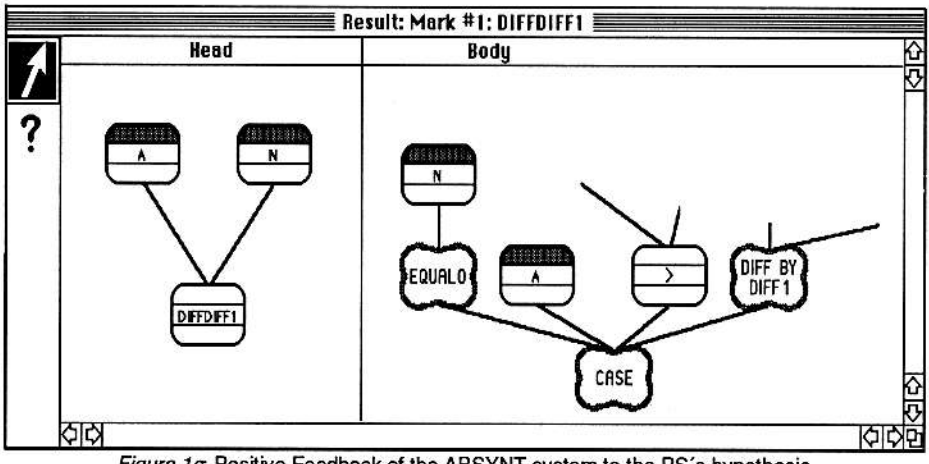


Figure 1c: Positive Feedback of the ABSYNT system to the PS's hypothesis

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. (Some of them are depicted on the left of Figure 2, see below.) Goal nodes represent more abstract plan fragments which may be implemented in several ways by implementation nodes or subtrees of implementation nodes. Visually, goal nodes have a different shape and no iconic internal structure. In Figure 1a, "EQUAL 0" 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 learner can see this description as well as a verbal description by clicking onto the node with the question mark tool.) For example, the "EQUAL 0" node represents the goal to test if a number is equal to 0. (Formally: "goal EQUAL 0 (number n) bool: that bool x: x = (n = 0)": This is the goal to determine for a number n that boolean value which results from evaluating "n = 0".) 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": This is the goal to determine for n condition-expression-pairs p_i, a_i that value which results from evaluating the first expression from left to right which condition is true.)

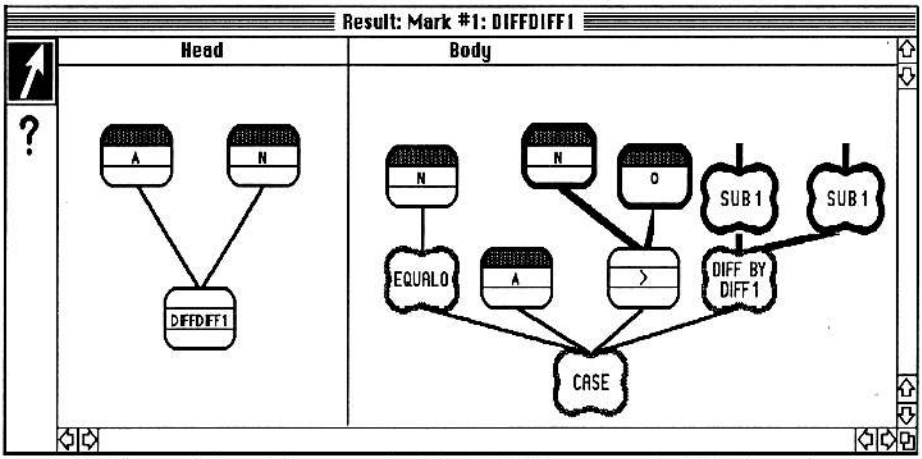


Figure 1d: Completion proposals of the ABSYNT system on the PS's demand

The ABSYNT goal nodes are based on a task analysis which applies the *transformation* approach developed in the Munich CIP Project [6, 32]. 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 solution proposal is just being created for the ABSYNT programming task "diff by diff 1": "Create a program that subtracts a natural number from a number. The subtraction operator can only be used with '1' as its second input." In the not yet finished proposal shown in Figure 1a, there are completely unspecified nodes (shaded areas) and partially unspecified (yet unnamed) nodes. As Figure 1a also shows, goal nodes and implementation nodes can be mixed ("mixed trees") within a proposal. The solution proposal in Figure 1a means:

> If the value of the parameter N is equal to zero,
> then the value of DIFFDIFF1 is the value of the parameter A,
> else
> if the value of a yet unspecified parameter is greater than
> the value of a yet unspecified constant,
> then the value of DIFFDIFF1 is obtained by realizing the goal "diff by diff 1"
> for a yet unspecified expression and
> the subtraction of 1 from a yet unspecified parameter.

In the *hypotheses testing environment* the learner may state hypotheses (bold parts of the program in Figure 1b) 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!". The system then analyzes the hypothesis. In Figure 1b the learner stated a hypothesis which covers a fragment of the proposal created so far for the "diff by diff 1" programming task. The hypothesis contains goal nodes and implementation nodes. The system recognizes the hypothesis as embeddable, indicating this by returning a copy of the hypothesis to the PS (Figure 1c). If this information is not sufficient for

resolving the impasse, the PS may ask the system for completion proposals at the open links. In Figure 1d, the PS asked for and received four completions (bold). Two of them are goal nodes (SUB1). As far as possible, the system tries to generate nodes which are already contained in the PS´s proposal. Internally, the system has created a complete solution but the PS always gets only minimal information. On the other hand, if the PS stated a hypothesis that cannot be confirmed, then the PS receives the message that the hypothesis cannot be completed to a solution known by the system.

The hypotheses testing environment 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 [4, 5, 10, 11], the SCENT advisor [16, 17], and the ELM system [40]. This is true also for the difference of ABSYNT and the visual data flow programming system "Function Machines" [12]. 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 PS and thereby provides a rich data source about the her or his 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 wrong solution proposals, 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 trees. In addition, a hypothesis may include only a part of the actual proposal. So the PS may obtain feedback whether she or 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)* [21]. 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. 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. EXPERT is able to analyze and to synthesize several millions of plans and solutions for the 42 tasks [22, 27]. We think that such a large solution space is necessary because we observed that especially novices often construct unusual solutions due to local repairs.

The goal decomposition done by the planning rules follows the CIP transformation approach mentioned earlier. So the goals and subgoals which are contained in the planning rules, and which correspond to the ABSYNT goal nodes useable by the learner, are based on the CIP approach. The CIP approach ensures that a solution can be derived to a given task that is correct with respect to the task description. So systematical, *derivational* programming is possible. The **appendix** illustrates how the solution to a given task can be derived, based on the CIP approach, and how the corresponding terms are represented in ABSYNT.

Empirical work

As already indicated, working with goal nodes in ABSYNT should enable the learner to express ideas at very early phases of program development, i.e., the *deliberating* and *planning* phases, and get feedback about these ideas. This general hypothesis leads to several specific hypotheses which may be evaluated empirically:

- If the PS has no goal nodes to work with, he will verbalize his goals, but there are no directly corresponding actions possible. But if the PS works with goal nodes, then his verbalizations will correspond more closely to his actions (i.e., editing goal nodes).

- If the PS has goal nodes to work with, then the pauses where the PS thinks and talks without performing programming actions will tend to become shorter, because the PS is able to express his ideas and intentions directly in ABSYNT, even if they are yet vague.

• If the PS has goal nodes to work with, then he will do more hypotheses testing, especially with goal trees and mixed trees, because this way the PS sees whether he is "on the right track" at early stages of solution development.

• For the same reason, the number of corrections will be reduced: If the PS has no goal nodes to work with, then faulty solution approaches will not be revealed as early as with goal nodes. In addition, goal nodes make it easier to find bugs by narrowing hypotheses.

In a preliminary empirical investigation a single subject worked through the sequence of ABSYNT tasks, using the goal nodes. Up to now, we compared an about one hour portion of her protocol with the corresponding portion of another subject working through the same sequence without the goal nodes. The analyzed protocol included the tasks "diff by diff 1" described earlier, and the task "even": "Create a program that tests whether a number is even." With respect to the hypotheses stated above, we examined the number of goal verbalizations without corresponding programming actions, the number of long pauses (more than a minute) between two programming actions, the number of hypotheses tested, and the number of corrections. Concerning verbalizations, pauses, and corrections, we found no differences in the protocols, but more empirical analyses are needed. Concerning the number of hypotheses tested, the subject working without goal nodes tested two hypotheses. In contrast, the subject who had goal nodes tested 9 hypotheses, many of them refering to partial plans and mixed trees. Figure 2 shows an example from the "even" task. The subject states the general solution plan that a number is even if its integer division by 2 leaves a rest that is equal to 0. The subject subsequently tested this plan as a hypothesis, so she knew early that she pursued a correct plan. Furthermore, at the task "diff by diff 1" the subject created the two last plans shown in the appendix (steps 11 and 12).

Figure 2 also shows in what sense the ABSYNT goal nodes support not only the problem solving phase of planning, but also the deliberating phase. "MOD2" in Figure 2 is a subtask which has to be planned and implemented as a separate recursive program. So the subject's decision for the goal node "MOD2" is considered as a result of deliberating. Moreover, ABSYNT would support deliberating by proposing the "MOD2" node if the subject tested a hypothesis containing the "EQUAL 0" node and the link leaving it.

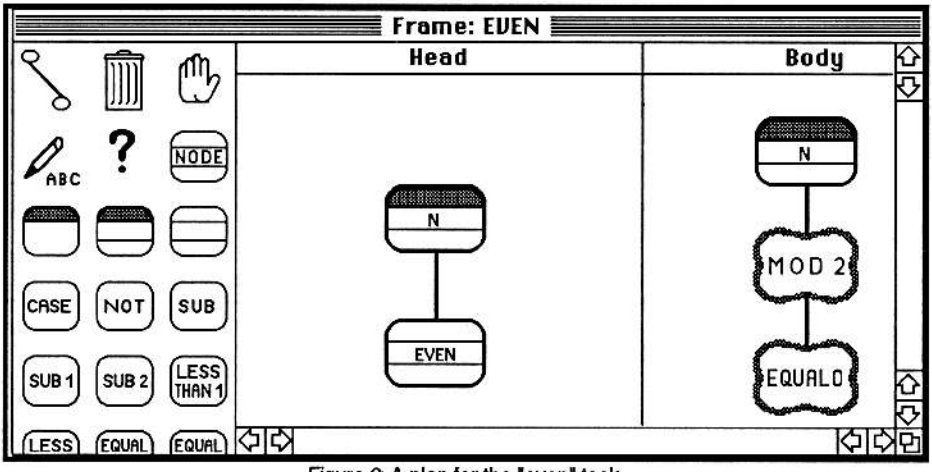


Figure 2: A plan for the "even" task

Making help adaptive to the problem solver's actual knowledge

The completions shown in Figure 1d (bold program fragments) were generated by the GMR EXPERT rules described above. 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 several alternatives for the "SUB1" goal node in Figure 1d. Thus the problem is to *select* the most appropriate completion proposal. So a model of the learner's actual knowledge state is needed, as recommended by the design criteria stated earlier.
We developed such a model which we call a *State Model* since it represents the successive knowledge state 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 means that the State Model tries to offer a completion proposal which is based on a rule contained in the State Model. So the learner's surprise to feedback and completion proposals should be minimized. 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. The State Model is described in detail in [25, 26].
The State Model is designed to be consistent with the ISP-DL Theory. Thus it contains acquired knowledge (micro rules) and optimized knowledge (schemas, cases). But it does not contain weak heuristics, control processes, and knowledge acquisition processes. This is the function of a *Process Model* [34, 35] which is developed and run offline. It provides the hypothetical reasons for the knowledge state changes represented in the State Model, and thereby is intended to bridge the gap between the State Model and the ISP-DL Theory.

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 level 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 primarily 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 PS's point of view, the benefit of *planning* with goal nodes is that hypotheses testing is possible already at the planning stage, and at *very early stages* of solution development in general. So the PS will get information whether she or he is "on the right track" before starting with the implementation. In a preliminary empirical investigation we investigated some empirical hypotheses stemming from this general hypothesis, indicating that hypotheses testing based on partial plans and (sometimes) on mixed trees was used frequently.
• 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. Thus it will

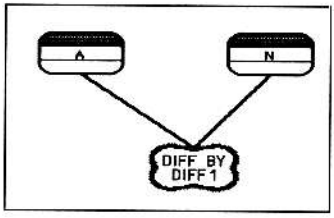
become possible to base an automatic online analysis of the PS's actions, which is necessary for the State Model, on data about *planning* as well. So one aspect of our future work is to extend the State Model accordingly.

• Finally, from a help system design point of view, the benefit is that in addition to hypotheses testing with goal nodes and goal completions, it will be possible to offer *planning rules* as help to the learner. So it should be possible to enable *derivational* programming by offering the CIP transformation rules to the learner. In addition, the CIP rules may be used to offer *explanations* for the system's completion proposals to the PS. This will be a second aspect of our work.

Appendix: Program Transformation Approach and Corresponding ABSYNT Constructs

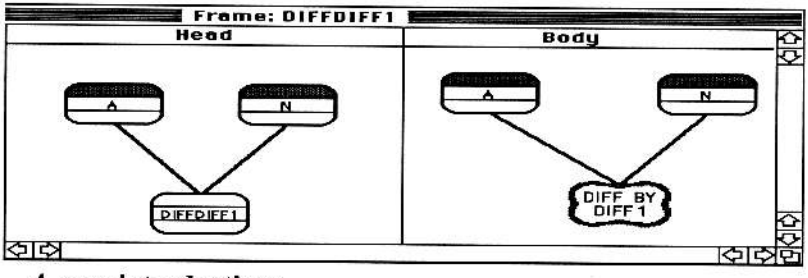
The transformation steps are explained at the end of the appendix.

1. task: "diff by diff 1"
"subtract a natural number from a number using only sub1"



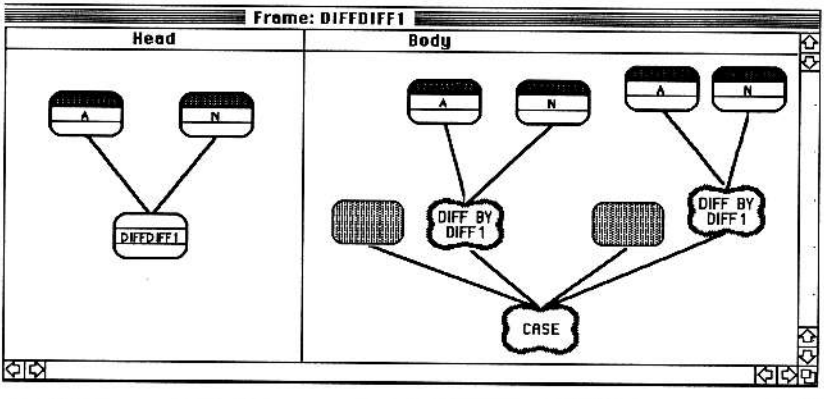
2. task specification:
that num x: x = a - n

3. function scheme:
funct diffdiff1 **(num** a, **nat** n) **num:**
that num x: x = a - n



4. case introduction:
funct diffdiff1 **(num** a, **nat** n) **num:**
if B1 **then that num** x: x = a - n
...
if Bm **then that num** x: x = a - n **fi** ... **fi**

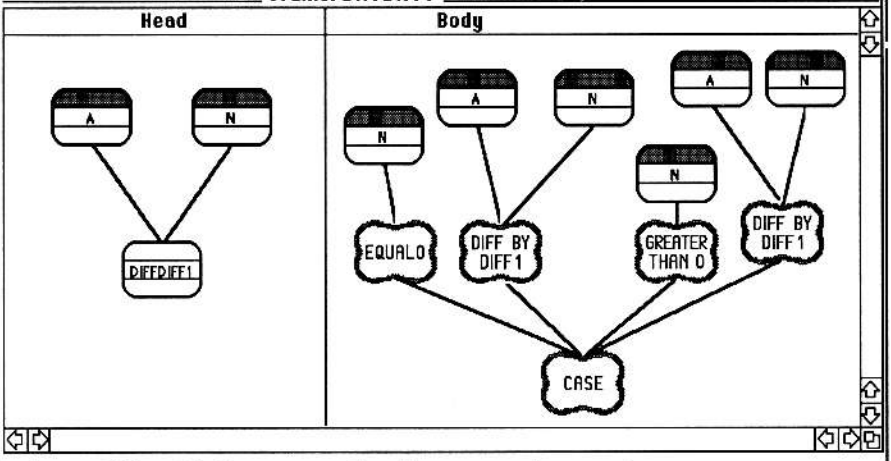
where: m = 2



5. predicate introduction:

funct diffdiff1 (**num** a, **nat** n) **num**:
if n = 0 **then that num** x: x = a - n
if n > 0 **then that num** x: x = a - n

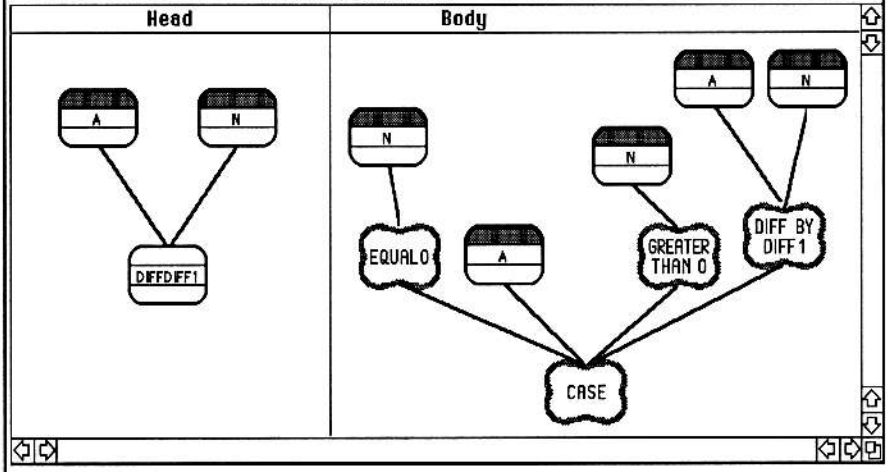
Frame: DIFFDIFF1



6. conditional inference under the constraint n = 0:

funct diffdiff1 (**num** a, **nat** n) **num**:
if n = 0 **then** a
if n > 0 **then that num** x: x = a - n

Frame: DIFFDIFF1



7. conditional inference under the constraint n > 0:

funct diffdiff1 (**num** a, **nat** n) **num**:
if n = 0 **then** a
if n > 0 **then that num** x: **exists num** x': [a - (n - 1) = x'] ∧ [x' - 1 = x]
(There is no ABSYNT representation for this step.)

8. choice and quantification:

applying the CIP-rule "choice and quantification"

that n x: exists m y:(P(y) ∧ f(y) = x)
Δ
‖ | Application condition:
‖_____| "=" is an
‖ | equivalence relation
‖ | of sort m
V
f(that m y: P(y))

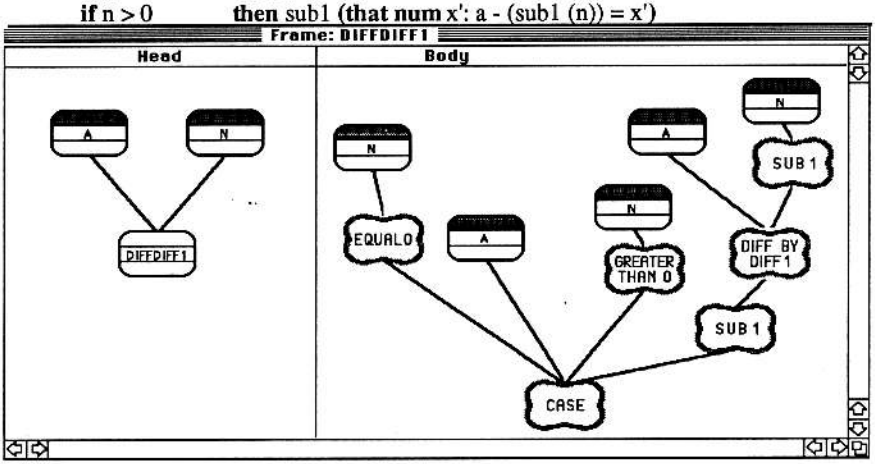
with bindings:

rule / derivation
x / x
y / x'
P(y) / a - (n - 1) = x' or
a - (sub1 (n)) = x'
f(y) / x' - 1 or sub1(x')

gives the expression:

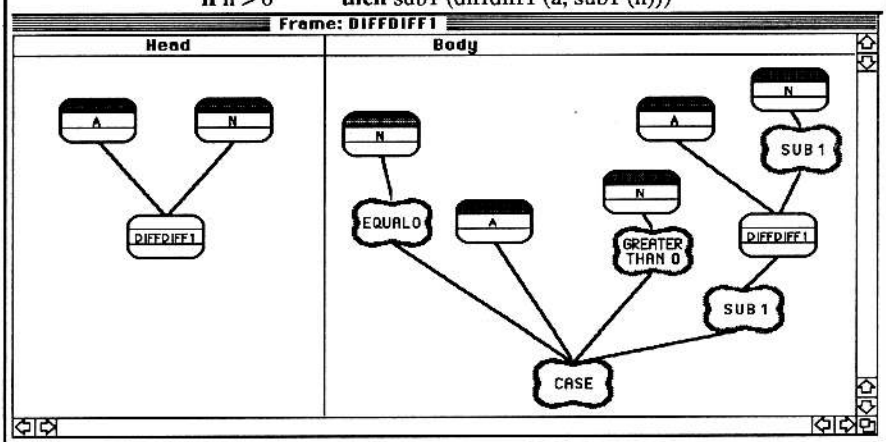
funct diffdiff1 **(num** a, **nat** n**) num:**
if n = 0 **then** a
if n > 0 **then** sub1 **(that num** x': a - (sub1 (n)) = x')

(specification of subtask)

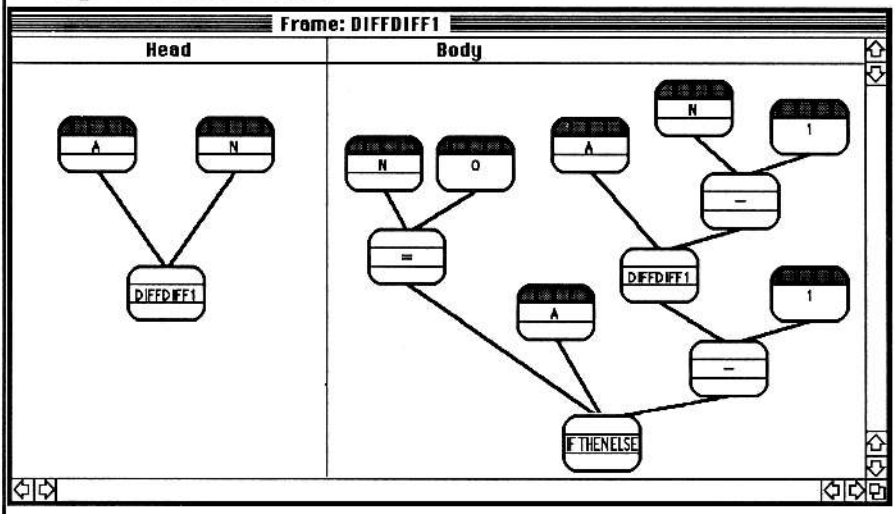


9. folding:

funct diffdiff1 **(num** a, **nat** n**) num:**
if n = 0 **then** a
if n > 0 **then** sub1 (diffdiff1 (a, sub1 (n)))



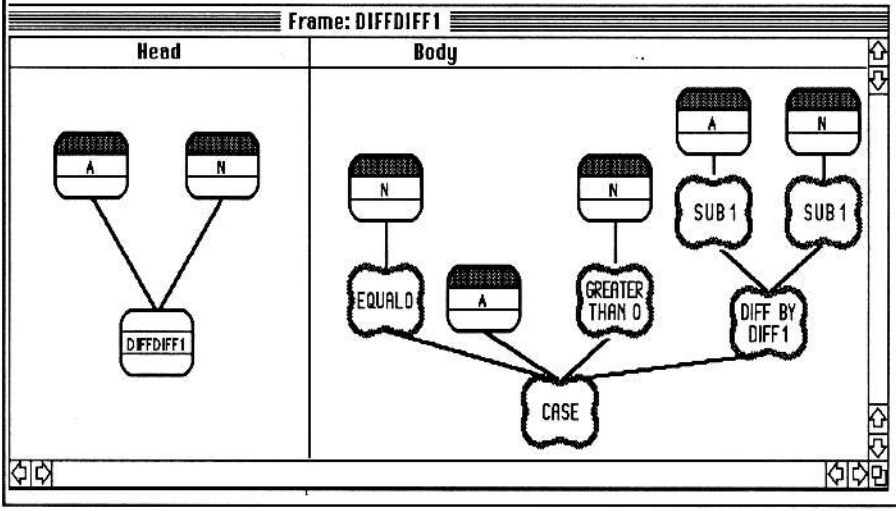
10. final recursive solution, containing only ABSYNT implementation nodes:



In each step of the program transformation process, there are several alternatives to continue. For example, the next two figures show a second way to apply the conditional inference under the constraint n > 0. The corresponding folding step leads to a tail recursive program solution.

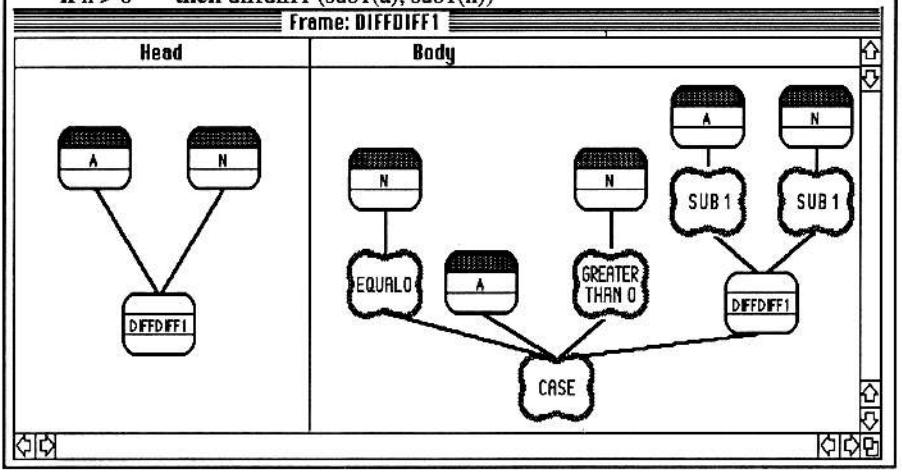
11. conditional inference under the constraint n > 0 :

funct diffdiff1 (**num** a, **nat** n) **num:**
if n = 0 **then** a
if n > 0 **then that num** x: sub1(a) - sub1(n) = x



12. folding:

funct diffdiff1 (**num** a, **nat** n) **num:**
if n = 0 **then** a
if n > 0 **then** diffdiff1 (sub1(a), sub1(n))



Comments to the appendix:

1) Informal specification of the task
2) Formal specification of the task: num is the sort of the result variable x; the specification is represented by a goal node in ABSYNT
3) The specification of the task has become the body of the two-parameter function diffdiff1.
4) We think that the problem can be solved by splitting it into subproblems: case analysis; for each case we retain the original task specification "that num x: x = a - n"
5) According to a corresponding CIP-rule we introduce predicates subject to some constraints: no gaps or overlaps in the domain of the function etc.; for demonstration purposes we prefer to represent the predicates and functions by goal nodes even if they could be implemented in one step by runnable ABSYNT operators.
6) Subject to the condition that some predicate is true we can try some conditional inferences; under the constraint n = 0 we are able to simplify the task specification to the value of "a".
7) Under the assumption that n > 0 we are allowed to specify a subtask "exists num x': [a - (n-1) = x']". To bridge the gap to the old task specification "that num x: x = a - n" we have to subtract: [x' - 1 = x]
8) The second if-clause in 7 matches with the left hand side of the CIP-rule "choice and quantification" (shown); the parameters in the rule get some bindings (shown on the right side of 8)
9) To the second if-clause in 8, we can apply the CIP "folding"-rule (not shown) which allows to substitute under certain conditions the task specification of the subtask "that num x': a - (sub1 (n)) = x' " by the recursive call "diffdiff1(a, sub1 (n))".
10) All goal nodes are substituted by runnable ABSYNT operators.
11)-12) An alternative derivation is shown here.

References

[1] Anderson, J.R., The Architecture of Cognition. Cambridge: Harvard University Press, 1983

[2] Anderson, J.R., Knowledge Compilation: The General Learning Mechanism, in Michalski, R.S.; Carbonell, J.G.; Mitchell, T.M.(eds), Machine Learning, Vol. II. Los Altos: Kaufman, 1986, 289-310

[3] Anderson, J.R., A Theory of the Origins of Human Knowledge. Artificial Intelligence, 40, 1989, 313-351

[4] Anderson, J.R., Conrad, F.G., Corbett, A.T., Skill Acquisition and the LISP Tutor, Cognitive Science, 1989, 13, 467-505

[5] Anderson, J.R., Swarecki, E., The Automated Tutoring of Introductory Computer Programming, Communications of the ACM, 1986, 29, 842-849

[6] Bauer, F.L., Ehler, H., Horsch, A., Möller, B., Partsch, H., Paukner, O., Pepper, P., The Munich Project CIP, Vol. II: The Program Transformation System CIP-S, Lecture Notes in Computer Science, Vol. 292, Berlin: Springer, 1987

[7] Bauer, F.L., Goos, G., Informatik (Vol. 1), Berlin: Springer, 1982 (3rd ed.)

[8] Brown, J.S., van Lehn, K., Repair Theory: A Generative Theory of Bugs in Procedural Skills, Cognitive Science, 1980, 4, 379-426

[9] Chang, S.K. (ed), Principles of Visual Programming Systems, Englewood Cliffs: Prentice Hall, 1990

[10] Corbett, A.T., Anderson, J.R., Patterson, E.J., Problem Compilation and Tutoring Flexibility in the LISP Tutor, Procedings of the 1st Int. Conf. on Intelligent Tutoring Systems ITS-88, Montreal, 423-429

[11] Corbett, A.T., Anderson, J.R., 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), Lecture Notes in Computer Science, Vol. 608, Berlin: Springer, 1992, 413-420

[12] Feuerzeig, W., Richards, J., Roberts, N., Function Machines - User Manual, BBN Laboratories, 1989

[13] Glinert, E.P., Nontextual Programming Environments, in Chang (ed), Principles of Visual Programming Systems, Englewood Cliffs: Prentice Hall, 1990, 144-230

[14] Gollwitzer, P.M., Action Phases and Mind-Sets, in: E.T. Higgins & R.M. Sorrentino (eds), Handbook of Motivation and Cognition: Foundations of Social Behavior, 1990, Vol.2, 53-92

[15] Gollwitzer, P.M., Abwägen und Planen, Göttingen, Toronto: Verlag für Psychologie, 1991

[16] Greer, J., Granularity and Context in Learning, University of Saskatchewan, Saskatoon, Canada, 1992, Invited Talk at the 2nd Int. Conference ITS 92, Montreal, June 10 - 12, 1992

[17] Greer, J., McCalla, G.I., Mark, M.A., Incorporating Granularity-Based Recognition into SCENT, Proceedings 4th Int. Conference on Artificial Intelligence and Education, Amsterdam: IOS, 1989

[18] Laird, J.E., Rosenbloom, P.S., Newell, A., Universal Subgoaling and Chunking. The Automatic Generation and Learning of Goal Hierarchies, Boston: Kluwer, 1986

[19] Laird, J.E., Rosenbloom, P.S., Newell, A., SOAR: An Architecture for General Intelligence, Artificial Intelligence, 1987, 33, 1-64

[20] Larkin, J.H., Simon, H.A., Why a Diagram is (Sometimes) Worth Ten Thousand Words. Cognitive Science, 1987, 11, 65-99

[21] Levi G., Sirovich, F., Generalized And/Or-Graphs. Artificial Intelligence, 1976, 7, 243-259

[22] Möbus, C., 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, Proceedings of the IFIP TC3 Int. Conf. on Advanced Research on Computers in Education Tokyo, Japan, 18-20 July, 1990, Elsevier Science Publ. (North-Holland), 1991, 137-144

[23] Möbus, C., Pitschke, K., Schröder, O., 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, Lecture Notes in Computer Science, Vol. 608, Berlin: Springer, 1992, 294-301

[24] Möbus, C., Schröder, O., 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, Lecture Notes in Computer Science, Vol. 439, Berlin: Springer, 1990, 47-81

[25] Möbus, C., Schröder, O., Thole, H.-J., A Model of the Acquisition and Improvement of Domain Knowledge for Functional Programming, J. of Artificial Intelligence in Education, 1992, 3(4), 449 - 476

[26] Möbus, C., Schröder, O., Thole, H.-J., Diagnosing and Evaluating the Acquisition Process of Problem Solving Schemata in the Domain of Functional Programming, to appear in G. McCalla (ed), Student Modelling: The Key to Individualized Knowledge-Based Instruction, in press

[27] Möbus, C., Thole, H.-J., 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, Lecture Notes in Computer Science, Vol. 438, Berlin: Springer, 1990, 36-49

[28] Möbus, C., Thole, H.-J., Schröder, O., Interactive Support of Planning in a Functional, Visual Programming Language, to appear in Proceedings of the 5th Int. Conf. on Artificial Intelligence in Education AIED 93, in press

[29] Neves, D.M., Anderson, J.R., Knowledge Compilation: Mechanisms for the Automatization of, Cognitive Skills, in Anderson, J.R. (ed), Cognitive Skills and their Acquisition. Hillsdale, N.J.: Erlbaum, 1981, 57-84

[30] Newell, A., The Knowledge Level. Artificial Intelligence, 1982, 18, 87-127

[31] Newell, A., Unified Theories of Cognition, Cambridge: Harvard University Press, 1990

[32] Partsch, H.A., Specification and Transformation of Programs: A Formal Approach to Software Development, Berlin: Springer, 1990

[33] Rosenbloom, P.S., Laird, J.E., Newell, A., McCarl, R., A Preliminary Analysis of the SOAR Architecture as a Basis for General Intelligence, Artificial Intelligence, 1991, 47, 289-305

[34] Schröder, O., A Model of the Acquisition of Rule Knowledge with Visual Helps: The Operational Knowledge for a Functional, Visual Programming Language, 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, Lecture Notes in Computer Science, Vol. 438, Berlin: Springer, 1990, 142-157

[35] Schröder, O., Möbus, C., Zur Modellierung des hilfegeleiteten Wissenserwerbs beim Problemlösen, in K. Reiss, M. Reiss, H. Spandl (Hrsg), Maschinelles Lernen - Modellierung von Lernen mit Maschinen, Berlin: Springer, 1992, 23-62

[36] Schröder, O., Möbus, C., Pitschke, K., Designing Help for Viewpoint Centered Planning of Petri nets, to appear in Proceedings of the 5th Int. Conf. on Artificial Intelligence in Education AIED 93, in press

[37] Van Lehn, K., Toward a Theory of Impasse-Driven Learning. In: Mandl, H.; Lesgold, A. (eds): Learning Issues for Intelligent Tutoring Systems. New York: Springer, 1988, 19-41

[38] Van Lehn, K., Mind Bugs: The Origins of Procedural Misconceptions, Cambridge: MIT Press, 1990

[39] Van Lehn, K., Rule Acquisition Events in the Discovery of Problem Solving Strategies, Cognitive Science, 1991, 15, 1-47

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

[41] Wolff, J.G., Cognitive Development as Optimisation, in Bole, L. (ed), Computational Models of Learning. Berlin: Springer, 1987, 161-205

[42] Wolff, J.G., Towards a Theory of Cognition and Computing, Chichester: Ellis Horwood, 1991

Simulation-Based Experiential Learning

Edited by

Douglas M. Towne

Behavioral Technology Laboratories
University of Southern California
250 N. Harbor Drive, Suite 309
Redondo Beach, CA 90277, USA

Ton de Jong

Department of Education
University of Twente, PO Box 217
7500 AE Enschede, The Netherlands

Hans Spada

Department of Psychology
University of Freiburg
D-79085 Freiburg, Germany

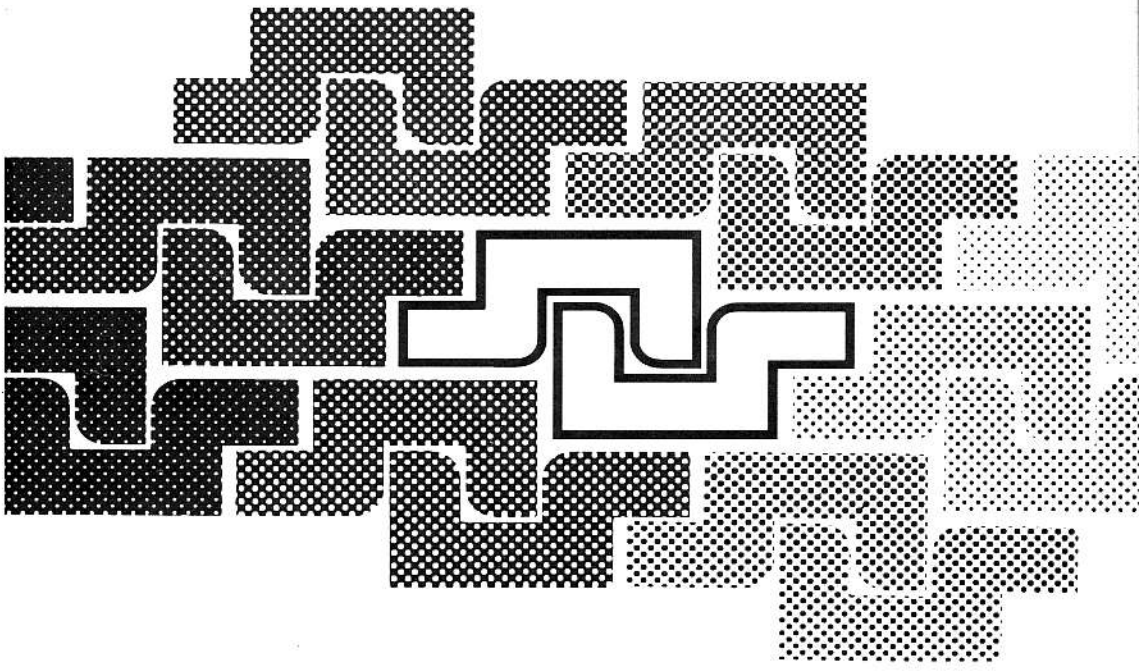
Springer-Verlag
Berlin Heidelberg New York London Paris Tokyo
Hong Kong Barcelona Budapest
Published in cooperation with NATO Scientific Affairs Division

Proceedings of the NATO Advanced Research Workshop on The Use of Computer Models for Explication, Analysis and Experiential Learning, held in Bonas, France, October 12–14, 1992

CR Subject Classification (1991): K.3, J.4, I.2, I.6

ISBN 3-540-57276-7 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-57276-7 Springer-Verlag New York Berlin Heidelberg

Typesetting: Camera-ready by authors/editors
45/3140 - 5 4 3 2 1 0 - Printed on acid-free paper



Simulation-Based Experiential Learning

Edited by
Douglas M. Towne Ton de Jong Hans Spada

NATO ASI Series

Series F: Computer and Systems Sciences, Vol. 122