Diagnosing and Evaluating the Acquisition Process of Problem Solving Schemata in the Domain of Functional Programming

Claus Möbus, Olaf Schröder, and Heinz-Jürgen Thole*

Department of Computational Science, University of Oldenburg, Oldenburg, Germany

Abstract: This paper describes an approach to model students' knowledge growth from novice to expert within the framework of a help system, ABSYNT, in the domain of functional programming. The help system has expert knowledge about a large solution space. On the other hand, in order to provide learner-centered help there is a model of the student's actual state of domain knowledge. The model is continuously updated based on the learner's actions. It distinguishes between newly acquired and improved knowledge. *Newly acquired knowledge* is represented by augmenting the model with rules from the expert knowledge base. Although they are expert rules, only rules able to explain the student's action sequences are incorporated in the model. *Knowledge improvement* is represented by rule composition. This allows the prediction of various knowledge acquisition phenomena, like performance speedup and a decrease of verbalizations.

In this way, the knowledge contained in the model is partially ordered from general rules to more specific schemas for solution fragments to specific cases (= example solutions). The model construction is implemented but not yet actually used for help generation within the help system. This paper focuses on knowledge diagnosis as accomplished by the model, and on an empirical analysis of some of its predictions.

Keywords: knowledge acquisition, knowledge optimization, schema identification, empirical validation of student models, analysis of time-based and correction-based data

---

* We thank Jörg Folckers for reimplementing ABSYNT in LPA-PROLOG for the Macintosh computer. Now we can switch off our LISP machine.

1 Introduction

The problem of student modelling has become an important research topic especially within the context of help and tutoring systems [5, 9, 18, 26, 53, 54, 64] because the design of such systems raises questions like: Which order is the best for a set of tasks to be worked on? Why is information useless to one person and helpful to another? How is help material to be designed? Advance in answering these questions seems to be possible only if the actual knowledge state of the learner can be diagnosed *online* in an efficient and valid way. This is difficult [50, 51] but necessary for a system in order to react adequately to the student's activities. Furthermore, it has been well recognized that progress in student modelling depends much on understanding what the student is doing (and why). Thus, detailed assumptions about problem solving, knowledge representation and acquisition processes are needed.

We face the student modelling problem within the context of a help system in the domain of functional programming: The ABSYNT Problem Solving Monitor. ABSYNT ("Abstract Syntax Trees") is a functional visual programming language designed to support the acquisition of basic functional programming knowledge. The ABSYNT Problem Solving Monitor provides help and proposals for the student while constructing ABSYNT programs to solve given tasks. In order to make the system's actions adaptive to the student, we model the growth of the student's knowledge state. Our basic approach rests on three principles:

- To try to "understand what the student is doing", and why. This amounts to constructing a *theoretical framework* which is powerful enough to describe the continuous stream of hypothetical problem solving, knowledge acquisition and utilization events, and to explain the stream of observable actions and verbalizations of the student.
- To use a subset of this theoretical framework in order to construct a student model containing the actual hypothetical state of domain knowledge of the student. This *state model* must be (and can be) simpler than the theoretical framework because its job is *efficient online diagnosis of domain knowledge* based on the computer-assessable data provided by the student's interactions with the system.
- To fill the gap between the theoretical framework and the state model by constructing an offline model of knowledge acquisition, knowledge modification, and problem solving processes. This *process model* provides hypothetical *reasons* for the changing knowledge states as represented in the state model.

In accordance with these principles, we pursue a three-level approach:

- A theoretical framework of problem solving and learning serves as a base for interpreting and understanding the student's actions and verbalizations. We call this framework *ISP-DL Theory* (Impasse - Success - Problem - Solving - Driven Learning Theory).

- An *internal model* (*IM*) diagnoses the actual domain knowledge of the learner at different states in the knowledge acquisition process (*state model*). It is designed to be an integrated part of the help system ("internal" to it) in order to provide user-centered feedback.
- An *external model* (*EM*) is designed to simulate the knowledge acquisition *processes* of learners at a level of detail not available to the IM (for example, including verbalizations). Thus the EM is not part of the help system ("external" to it) but supports the design of the IM.

Thus ISP-DL Theory, IM, and EM are designed to be mutually consistent but serve different purposes. This paper is concerned with the IM. It is organized as follows: First we will briefly describe the ISP-DL Theory, our help system, the ABSYNT problem solving monitor, and the domain of functional programming knowledge as incorporated in ABSYNT. Then the IM is described and illustrated in some detail. Empirical predictions and a first evaluation are presented. Finally we will discuss some possible extensions and the role of the IM for adaptive help generation.

2 The ISP-DL Knowledge Acquisition Theory

As indicated, the ISP-DL Theory is intended to describe the continuous flow of problem solving and learning of the student as it occurs in a sequence of, for example, programming sessions. In our view, existing approaches touch upon main aspects of this process but do not cover all of them. Consequently, the ISP-DL Theory is an attempt to integrate several approaches. Before describing it, we will briefly discuss three theoretical approaches relevant here:

- In VanLehn's [56, 58, 60] theory of Impasse Driven Learning, the concept of an impasse is of central importance to the acquisition of new knowledge. Roughly, an impasse is a situation where "the architecture cannot decide what to do next given the knowledge and the situation that are its current focus of attention" [60, p. 19]. Impasses trigger problem solving processes which may lead to new information. Thus, impasses are an important source for the acquisition of new knowledge, though probably not the only one [57, 60]. Impasses are also situations where the learner is likely to actively look for and to accept *help* [56]. There is also empirical evidence that uncertainty leads to active search for information [30]. But problem solving or trying to understand remedial information might as well lead to secondary impasses [10].

  The idea of impasse-driven learning is also found elsewhere. As an example from machine learning, Prodigy [11, 35] acquires new domain knowledge and new heuristics in response to noticing differences between expected and obtained outcomes. As an example from memory research, scripts may be augmented with information about exceptions in response to mispredicted events [31, 47]. Refining hypotheses in the context of concept learning [15] may be considered another instance.

Impasse Driven Learning Theory is concerned about *conditions* for problem solving, using help, and thereby acquiring new knowledge. It is not concerned about optimizing knowledge already acquired. "Knowledge compilation ... is not the kind of learning that the theory describes" [56, p. 32]. Thus Impasse Driven Learning Theory covers an important part of the processes we are interested in, but not all of them.

- In SOAR [28, 29, 45] the concept of impasse driven learning is elaborated by different types of impasses and weak heuristics performed in response to them. Impasses trigger the creation of subgoals and heuristic search in corresponding problem spaces. If a solution is found, a chunk is created acting as a new operator in the original problem space.

  In SOAR all learning is triggered by impasses. But these impasses can be more fine-grained than in VanLehn's theory. Since our intention is to describe and understand students' actions and verbalizations, we are interested in coarse-grained impasses corresponding to observable behavior. At this level of analysis, it seems questionable whether all knowledge acquisition events can reasonably be described as resulting from impasses [57, 60]. For example, existing knowledge may be deductively improved as a result of its successful application without changing the problem space.
- ACT* [1, 2, 4] focuses on the success-driven optimization of already existing knowledge by knowledge compilation but pays less attention to the problem of where new knowledge comes from. This is a main topic of PUPS [2-4] which provides mechanisms for the inductive acquisition of rules from the perception of causal relationships and from analogy. But conditions for knowledge acquisition events (like impasses) is less focused on.

We think that for our purposes it is necessary to cover problem solving, impasse-driven learning, and success-driven learning as well. Thus ISP-DL Theory incorporates the following aspects:

- The distinction of different problem solving phases [19, 20]: *deliberating* with the result of choosing a goal, *planning* a solution to it, *executing* the plan and *evaluating* the result.
- The *impasse-driven acquisition of new knowledge*. In response to impasses, the problem solver applies weak heuristics, like asking questions, looking for help, etc. [29, 56-58, 60]. Thus *new* knowledge may be *acquired*.
- The *success-driven improvement of acquired knowledge*. *Successfully used* knowledge is *improved* so it can be used more effectively. More specifically, by *rule composition*

[1, 2, 32, 43, 61], the number of control decisions and subgoals to be set is reduced. In our approach, composition is based on resolution and unfolding [24].

We describe the ISP-DL Theory by *hierarchical higher Petri nets* [25], though alternative modelling formalisms are possible, eg., *stream* communication [22]. Petri nets show temporal constraints on the order of processing steps more clearly than a purely verbal presentation. Thus they emphasize empirical predictions. The whole process is divided into 4 recursive subprocesses (*pages*): "Problem Processing", "Goal Processing", "Nonoperational Goal Processing" and "Operational Goal Processing" (Figures 1-4). *Places* (circles/ellipses) represent states (eg., the content of data memories); *transitions* (rectangles) represent events or process steps.

Places may contain tokens which represent mental objects (goals, memory traces, heuristics etc.) or real objects (eg. a solution or a behaviour protocol). Places can be marked with tags (*In* for entering, *Out* for exiting place, *FG* for global fusion set). An FG tagged place is common to several nets (eg. the Knowledge Base). Transitions can be tagged with HI (HI for hierarchical invocation transition). This means that the process is continued in the called subnet. The dotted boxes show which places are corresponding in the calling net and in the called net. Shaded transitions and places are taken into account by the IM (see below).

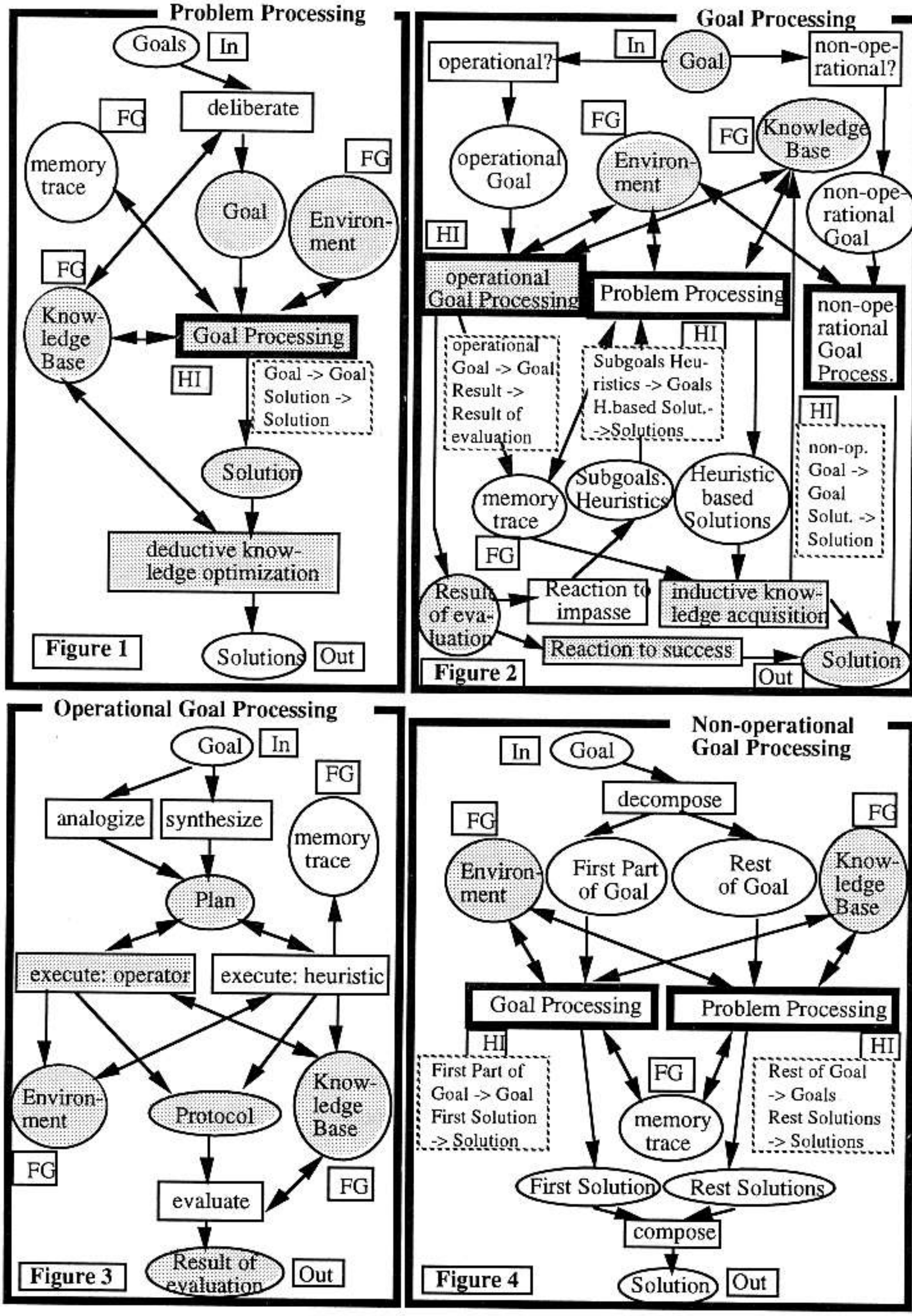
Problem Solving is started in the page "*Problem Processing*" (Figure 1). The problem solver (PS) strives for one goal to choose out of the set of goals: "*deliberate*".

A goal may be viewed as a set of facts about the environment which the problem solver wants to become true [44]. A goal can be expressed as a *predicative description* which is to be achieved by a problem solution. For example, the goal to create a program which tests if a natural number is even, "even(n)", can be expressed by the description: "funct even = (nat n) bool: exists ((nat k) 2 * k = n)". The "even" problem can be implemented by a function with the same name, one parameter "n" which has the type "*natural number*", the output type of the function is a *boolean* truth value, and the body of the function has to meet the declarative specification: "There exists a natural number k such that 2 * k = n". The goal is achieved when a program is created which satisfies this description.

The goal is processed in the page "*Goal Processing*" (Figure 2). If the PS comes up with a solution, the used knowledge is optimized: *deductive knowledge optimization*. When the PS encounters a similar problem, the solution time will be shorter. The net is left when there are no tokens in "*Goals*", "*Goal*" and "*Solutions*".

In the page "Goal Processing" (Figure 2) the PS checks whether his set of problem solving operators is sufficient for a solution: "*operational?*"/"*non-operational?*".

An operational goal is processed according to the page "*Operational Goal Processing*" (Figure 3). A plan is *synthesized* by applying problem solving operators, or it is created by



Figures 1 - 4: The ISP-DL theory of problem solving and learning *analogical* reasoning. The plan is a partially ordered sequence or hierarchy of domain specific problem solving goals (or of domain-unspecific heuristic goals, this will be explained in a moment). In either case, the goals in the plan are pursued by *executing* domain specific or heuristic operators. Execution leads to a problem solving *protocol* which is used in combination with the knowledge base to *evaluate* the outcome. The *result of the evaluation* generates an impasse or a success. The result of the evaluation is transferred back to the page *"Goal Processing"*.

Within the page "Operational Goal Processing", the cause of an impasse may be located at different points. For example, the "synthesize" process might create an insufficient plan because of missing planning knowledge or insufficient control knowledge to make a decision. When the PS executes the plan (possibly mentally, leading to a protocol of verbalizations) and evaluates it, the result is an impasse. Another possibility is that the "execution" process has insufficient operators or heuristics. Then evaluation of the protocol will also come up with an impasse.

The *reaction* of the PS to *success* is: leave *"Goal Processing"* with a *solution.* The reaction to an *impasse* is the creation of subgoals to use weak heuristics for problem solving. Now there is a recursive call to "Problem Processing". "Goal Processing" and "Operational Goal Processing" are called again. This time, within Operational Goal Processing a plan to use heuristics is synthesized and executed. (Simple examples for these weak heuristics are to use a dictionary, to find an expert to consult, and so on.) A memory trace of the situation which led to the impasse is kept. If the use of heuristics is successful, the result is twofold:

- The heuristically based solution is transferred back further to the instance of the page "Goal Processing" where the impasse arose. Now the impasse is solved. The obtained solution is related to the memory trace of the impasse situation. Thus within *"Goal Processing"* new *domain specific* problem solving operators are inductively *acquired.*
- The obtained heuristically based solution is transferred back to "Problem Processing". Thus in *"Problem Processing"* the *domain-unspecific* heuristic knowledge is deductively *optimized.* So next time the PS encounters an impasse, he or she will be more skilled and efficient in using a dictionary, finding someone to consult, etc.

When *"Processing"* (Figure 4), the problem is decomposed and the subsolutions are composed into a final solution.

It is possible and necessary to refine the theory's transitions and places. For our purpose this simple theory is sufficient. Important for the rest of the paper are the theoretically and empirically validated statements:

- *New knowledge is acquired only at impasse time after the successful application of weak heuristics and on the basis of memory traces.*

- *Information is helpful only in impasses and if it is synchronized with the knowledge state of the PS.*

3 The ABSYNT Problem Solving Monitor

The visual language ABSYNT is based on ideas stated in an introductory computer science textbook [8]. ABSYNT is a tree representation of pure LISP without the list data structure (but we currently incorporate it) 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 is described in [41]. The ABSYNT Problem Solving Monitor provides an *iconic programming environment* [12]. Its main components are a visual editor, trace, and a *help component: a hypotheses testing environment.*

In the editor (Figure 5) ABSYNT programs can be constructed. There is a head window and a body window. The left part of Figure 5 shows the tool bar of the editor: The bucket is for deleting nodes and links. The hand is for moving, the pen for naming, and the line for connecting nodes. Next, there is a constant, parameter and "higher" self-defined 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", "+", "-", "*", ...). 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. In Figure 5, a program is actually under development by a student. There are subtrees not yet linked and nodes not yet named or completely unspecified (shaded areas). The upper part of Figure 5 shows the Start window for calling programs. This is also where the visual trace starts if selected by the student. In the visual trace, each computational step is made visible by representing computation goals and results within the upper and lower region of operator nodes, and within the lower region of parameter nodes (see [38]).

In the *hypotheses testing environment* (Figure 6), the PS may state hypotheses (bold parts of the program in the upper worksheet in Figure 6) about the correctness of programs or parts thereof for given programming tasks. The hypothesis is: "It is possible to embed the boldly marked fragment of the program in a correct solution to the current task!". The PS then selects the current task from a menu, and the system analyzes the hypothesis. If the hypothesis can be confirmed, the PS is shown a copy of the hypothesis. If this information is not sufficient to resolve the impasse, the PS may ask for more information (completion proposals). If the hypothesis cannot be confirmed, the PS receives the message that the hypothesis cannot be completed to a solution known by the system.

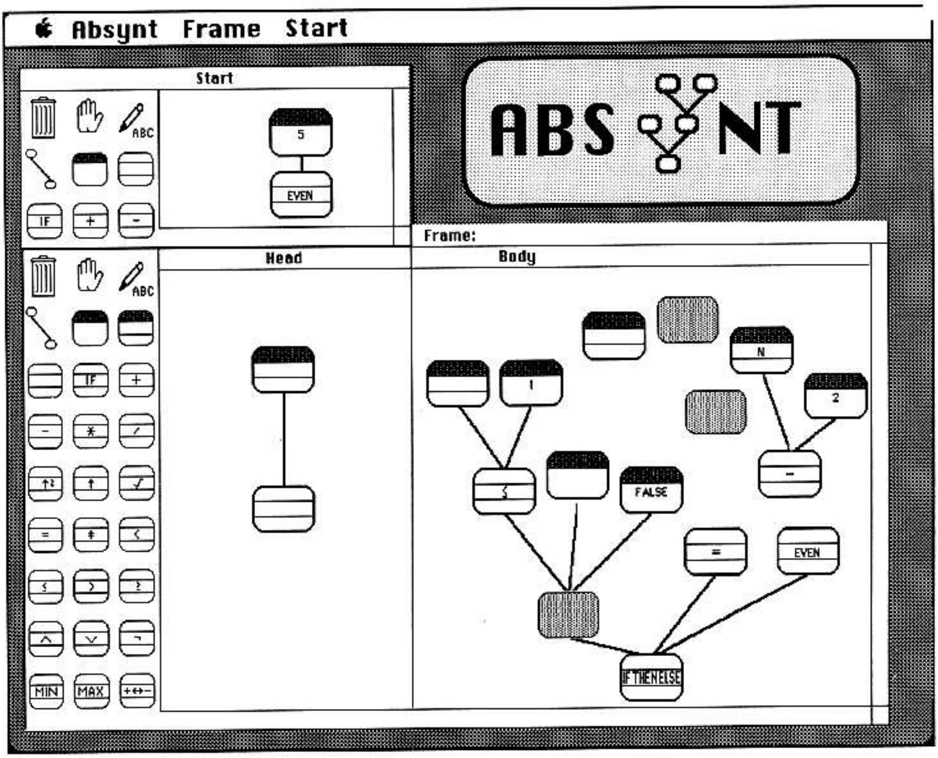


Figure 5: A snapshot of the visual editor of ABSYNT

The upper part of Figure 6 shows a proposed solution to the "even" problem just constructed by a student: "Construct a program that determines whether a number is even!" This solution does not terminate for odd arguments. In spite of that the *hypothesis* (bold program fragment in the upper part of Figure 6) is embeddable in a correct solution. So the hypothesis is returned as feedback to the student (thin program fragment in the middle part of Figure 6). The student then may ask for a completion proposal generated by the system. In the example the system completes the hypothesis successively with the constant "true" and with the "="-

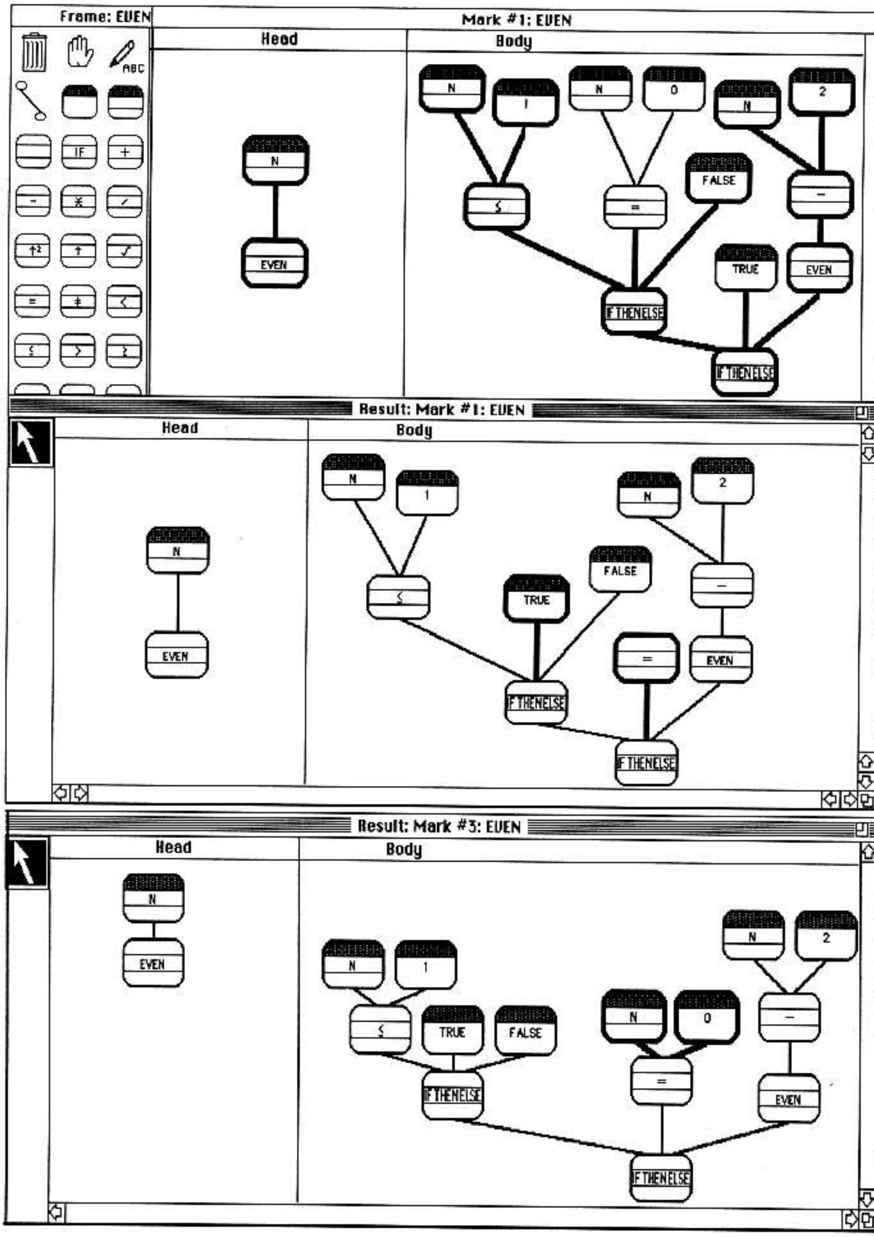


Figure 6: Snapshot of the ABSYNT hypotheses testing environment

operator (bold program fragments in the middle part of Figure 6). Internally, the system has generated a complete solution visible in the lower part of Figure 6. So the student's solution in the upper part of Figure 6 may be corrected by an interchange of program parts.

The hypotheses testing environment is the most significant aspect where the ABSYNT Problem Solving Monitor differs from other systems designed to support the acquisition of functional programming knowledge, like the LISP Tutor [6, 7, 14], the SCENT advisor [21, 33], and the ELM system [62]. One reason for the hypotheses testing approach is that in programming a bug usually *cannot be absolutely localized*, and there is a variety of ways to debug a wrong solution. 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 PS's knowledge state. Single subject sessions with the ABSYNT Problem Solving Monitor revealed that hypotheses testing was heavily used. It was almost the only means of debugging wrong solution proposals despite the fact that the subjects also had 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.

The answers to the learner's hypotheses are generated by rules defining a *goals-means-relation* A subset of these rules may be viewed as "pure" expert domain knowledge not influenced by learning. Thus we will call this set of rules EXPERT in the remainder of the paper. Currently, EXPERT contains about 650 rules and analyzes and synthesizes several million solutions for 40 tasks [36, 42]. One of them is the "even" task just introduced; more tasks will be presented later (see Figure 15). We think that such a large solution space is necessary because we observed that especially novices often construct unusual solutions due to local repairs. (This is exemplified by the clumsy-looking student proposal in the upper part of Figure 6.) Figure 7 depicts a hierarchy of types of rules in EXPERT. There are rules for programming (implementing), and rules for planning. The programming rules are split into rules implementing ABSYNT program heads (head rules), and rules implementing one ABSYNT node (node rules). The planning rules split into task plan rules and goal elaboration rules. Except for the task plan rules which will not be considered further, the following sections will provide definitions and examples of the different rule types.

The completions shown in the middle part of Figure 6 (bold program fragments) and the complete solution in the lower part of Figure 6 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. Thus the main function of the *IM* (internal student model), which rules are derived from EXPERT, is to *select* a completion from this set which is maximally *consistent* with the learner's current knowledge state. This should minimize the learner's surprise to feedback and completion proposals.

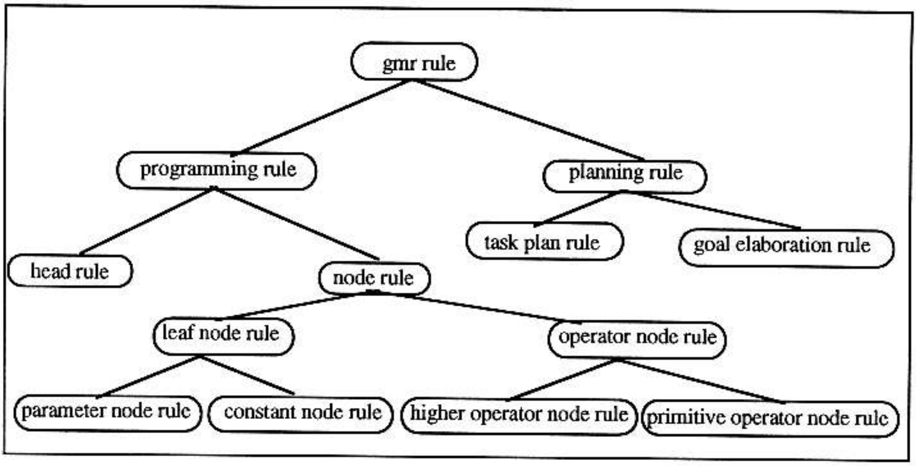


Figure 7: Hierarchy of types of rules in EXPERT

4 GMR Rules

This section describes the goals-means-relation GMR. The set of GMR rules may be split in two ways: *rule type* (simple, composed) vs. *database* of the rules (EXPERT, POSS, IM).

- As already indicated, there are three kinds of *simple rules*: *goal elaboration rules, rules implementing one ABSYNT node* (node rules), and *rules implementing ABSYNT program heads* (head rules).
- *Composite rules* are created by merging at least two successive rules parsing a solution. Composites may be produced from simple rules and composites. A composite is called a *schema* if it contains at least one pair of variables which can be bound to a goal tree and a corresponding ABSYNT program subtree. But if a composite is instantiated so that its variables can only be bound to node names or node values, then it is called a *case*.

The other way to partition the set GMR is the *data base* of the rules. EXPERT contains the ideal expert domain knowledge not changed by learning. So EXPERT contains only simple rules. The sets IM and POSS will be described below.

Figure 8 shows examples for simple rules depicted in their visual representations. Each rule has a *rule head* (left hand side, pointed to by the arrow) and a *rule body* (right hand side, where the arrow is pointing from). The rule head contains a *goals-means-pair* where the goal is contained in the ellipse and the means (implementation of the goal) is contained in the rectangle.

The rule body contains one goals-means-pair or a conjunction of pairs, or a primitive predicate (is_parm, is_const).

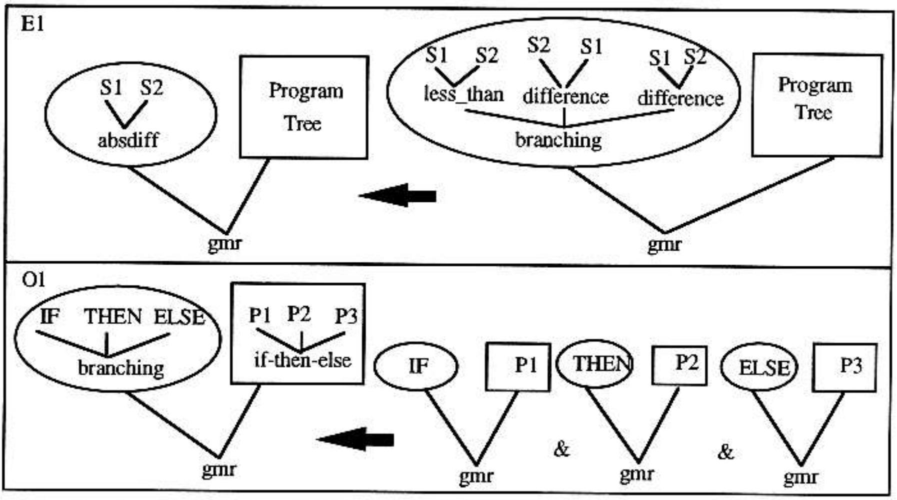


Figure 8: A goal elaboration rule (E1) and a rule (O1) implementing the ABSYNT node "if-then-else"

The first rule of Figure 8, E1, is a goal elaboration rule. It can be read:

If (*rule head*):

your main goal is "absdiff" with two subgoals S1 and S2,

then leave space for a program tree yet to be implemented, and (*rule body*):

If in the next planning step you create the new goal "branching" with the three subgoals "less_than (S1, S2)", "difference (S2, S1)", and "difference (S1,S2)",

then the program tree solving this new goal will also be the solution for the main goal"

O1 in Figure 8 is the "if-then-else" node rule (a primitive operator node rule), which is an example of a simple rule implementing one ABSYNT node (operator, parameter, or constant):

If (*rule head*):

your main goal is "branching" with three subgoals (IF, THEN, ELSE),

then *implement* an "if-then-else"-node (or "if-"-node) with three links leaving from its input, and leave space above these links for three program trees P1, P2, P3 yet to be implemented; and (*rule body*):

if in the next planning step you pursue the goal IF,

then its solution P1 will also be at P1 in the solution of the main goal, and
if in the next planning step you pursue the goal THEN,
then its solution P2 will also be at P2 in the solution of the main goal, and
if in the next planning step you pursue the goal ELSE,
then its solution P3 will also be at P3 in the solution of the main goal.

5 Composition of Rules

In our theory, composites represent improved sped-up knowledge. Together with the simple rules, they constitute a partial order from simple rules ("micro rules") to solution schemata to specific cases representing solution examples for tasks. In this section we will define rule composition.

If we view the rules as Horn clauses [27], then the composite RIJ of two rules RI and RJ can be described by the inference rule:

$$\frac{\text{RI: } (F \leftarrow P \,\&\, C) \quad \text{RJ: } (P' \leftarrow A)}{\text{RIJ: } (F \leftarrow A \,\&\, C)\sigma}$$

The two clauses above the line resolve to the resolvent below the line. A, C are conjunctions of atomic formulas. P, P', and F are atomic formulas. σ is the most general unifier of P and P'. RIJ is the result of unfolding RI and RJ - a sound operation [24].

For example we can compose the *schema* C7 (Figure 9) out of the set of simple rules {O1, O5, L1, L2}, where:

O1: gmr(branching(IF,THEN,ELSE),if-pop(P1,P2,P3)):-
gmr(IF,P1),gmr(THEN,P2),gmr(ELSE,P3).

O5: gmr(equal(S1,S2), eq-pop(P1,P2)):- gmr(S1,P1),gmr(S2,P2).L1:
gmr(parm(P), P-pl):- is_parm(P).

L2: gmr(const(C), C-cl):- is_const(C).

C7: gmr(branching(equal(parm(Y),const(C)),parm(X),ELSE),
if-pop(eq-pop(Y-pl,C-cl),X-pl,P)):-
is_parm(Y),is_const(C),is_parm(X),gmr(ELSE,P).

where:

if-pop	=	primitive ABSYNT operator "if-then-else" (or "if")
eq-pop	=	primitive ABSYNT operator "="
P-pl, X-pl, Y-pl	=	unnamed ABSYNT parameter leaves
C-cl	=	empty ABSYNT constant leaf

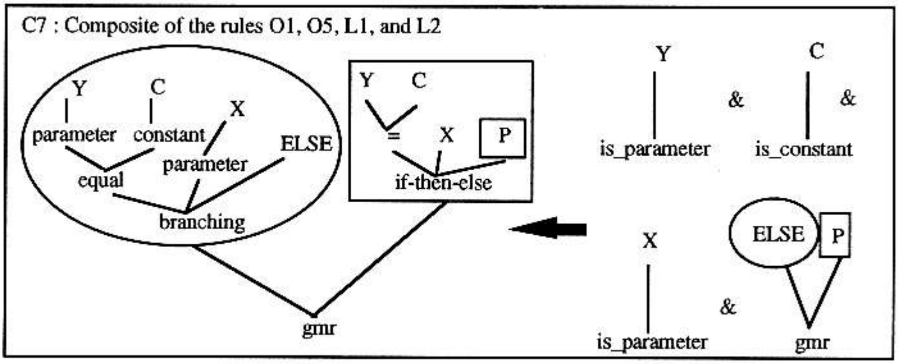


Figure 9: The composite C7

We also can describe the composition of node implementing rules RI and RJ with a shorthand notation:

$$RIJ = RI_k \bullet RJ$$

The index k denotes the place k in the goal tree of the head of RI. A place k is the k-th variable leaf numbered from left to right (eg.: $O1_3$ = ELSE). The semantics of "•" can be described in three steps. First, the variable in place k in the goal term in the head of RI is substituted by the goal term in the head of RJ. Second the call term P in the body of RI which contains the to be substituted variable unifies with the head of RJ and is replaced by the body of RJ. Third the unifier σ is applied to the term resulting from the second step, leading to the composed rule RIJ. Thus, the variables effected by the unification in step two are replaced by their bindings.

For example $O1_2 \bullet L1$ = gmr(branching(IF, parm(P), ELSE), if-pop(P1,P-pl,P3)):- gmr(IF,P1), is_parm(P), gmr(ELSE, P3). C7 can be composed from the rule set {O1, O5, L1, L2} in 16 different ways. Two possibilities are:

$$C7 = (O1_2 \bullet L1)_1 \bullet ((O5_2 \bullet L2)_1 \bullet L1)$$

$$C7 = (((O1_1 \bullet O5)_3 \bullet L1)_2 \bullet L2)_1 \bullet L1$$

6 Empirical Constraints of Simple Rules, Chains, Schemata and Cases

Rules, rule chains and schemata give rise to different *empirical predictions*. The purpose of this section is twofold:

- To introduce hypotheses about the application of novice and expert knowledge, viewed as simple GMR rules and composites. These hypotheses will be used in the Internal Model.
- To show which specific predictions follow from these hypotheses.

Any approach designed to represent changing knowledge states must mirror the shift from novice to expert. In general, novices work *sequentially*, set more subgoals, and need more control decisions, while experts work in *parallel*, set less subgoals, and need less control decisions [13, 16, 23, 52]. Here this difference is reflected in the partial order from simple rules to schemata to specific cases.

In order to demonstrate this difference, it is necessary to specify hypotheses about the problem solving behavior. According to the ISP-DL Theory, a plan is synthesized from a goal, and execution of operators leads to a protocol of actions and verbalizations (Figure 3). Thus with respect to the theory we make a distinction between the problem solving phases of *planning* and *execution*: A *plan synthesizer* or *"planner" synthesizes* plans, and an *operator executor* or *"coder" executes* operators to implement the plans. The coder has implementation knowledge ("programming rules" according to Figure 7) for implementing ABSYNT trees, but no planning knowledge. The coder also has very limited execution knowledge: pattern matching without unification (except for parameter and higher operator names, and constant values). More complex processes are left to the planner whose job is to guide the coder, based on domain specific planning knowledge and on weak heuristics (to be specified by the External Model, as stated earlier. For such a model in a related domain see [48, 49]).

For illustration of a hypothetical interaction sequence between planner and coder, we assume that the goal "branching (equal (parm(y), const(0)), parm(x), ELSE)" is to be implemented, and that the coder has knowledge about the set of simple GMR rules {O1, O5, L1, L2}. Figure 10 shows how the interaction might proceed: At time t_0, the planner delivers the goal. The coder has no rule for it so he rejects the goal. So the planner chops the goal into subgoals. Next, he may present the subgoal "parm(y)" to the coder. The coder now has a rule, L1, instantiates it to L1', and edits an ABSYNT parameter node with the name "y". Next, the planner delivers the subgoal "parm(x)". The coder uses L1 again, leading to the instantiation L1", and programs a parameter x. Then the planner comes up with "const(0)". The coder uses L2, applying L2' and programming a constant node 0. Next, the subgoal "equal(S1, S2)" is given. The planner instantiates O5 to O5' and creates a "=" node with two open links: their upper ends are shadows (place holders for nodes). After time t_j, the planner tells the coder that "equal(S1, S2)" has "parm(y)" as its first subgoal. So the coder connects the first input link of the "=" node to the parameter y. Next, the planner tells the coder that "equal(S1, S2)" has "const(0)" as its second subgoal, so the coder connects the second input link of the "=" node to the constant 0. Thus it

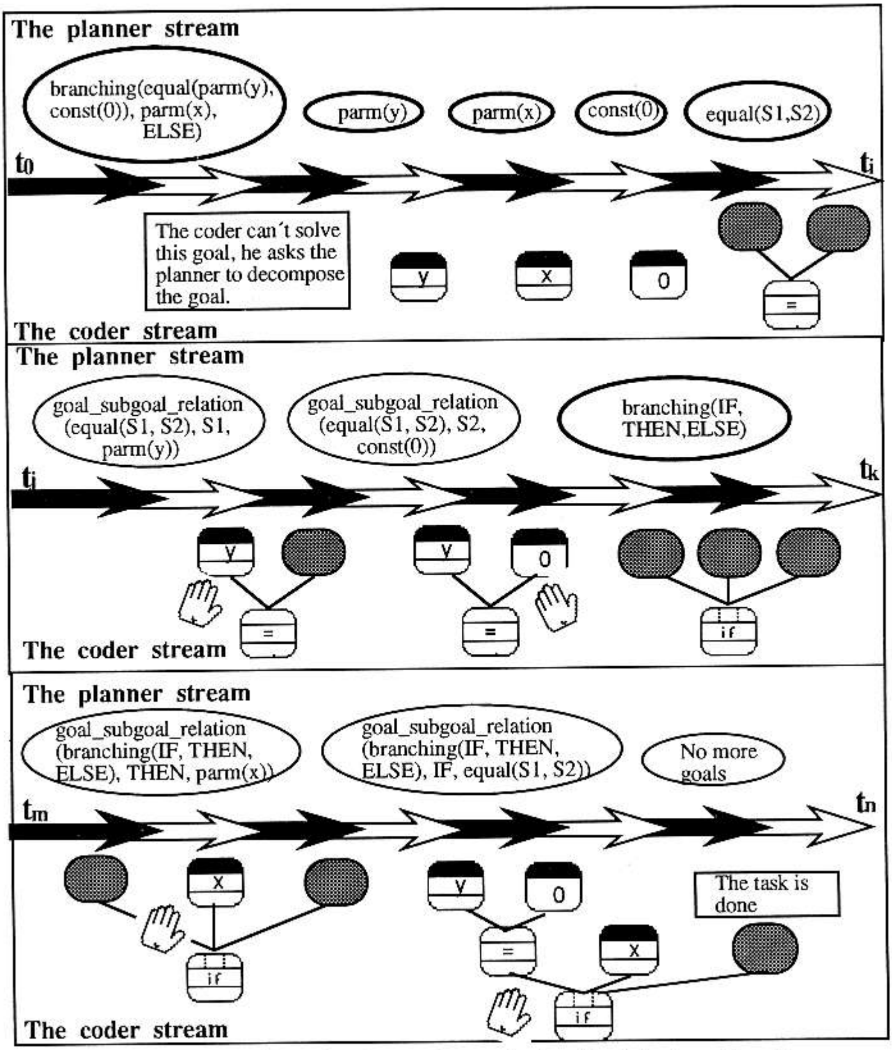


Figure 10: Sequence of interactions between planner and coder while solving the goal "branching (equal (parm(y), const(0)), parm(x), ELSE)" with the set {O1, O5, L1, L2} of simple rules

may be possible that the coder has to rearrange the position of the nodes and/or the orientation of the links. This is symbolized by the hand in Figure 10. Next, the planner comes up with the "branching(IF, THEN, ELSE)" subgoal. The coder implements it, instantiating O1 to O1'. After time t_m, the planner tells the coder that "branching(IF, THEN, ELSE)" has "parm(x)" as its second subgoal and "equal(S1, S2)" as its first subgoal. So the coder connects the second and first input link of the "if-then-else" node to the parameter x and to the "=" node, respectively. Again, the position of links and/or nodes on the screen may have to be rearranged. Now the goal is solved.

Thus the planner does not know about the coder's knowledge, and vice versa. There is no fixed order of application of GMR rules. The order solely depends on how the goals are delivered to the coder by the planner. In the example the coder created the sequence of rule instantiations (L1', L1", L2', O5', O1') depending on the goals delivered by the planner.

In contrast to this sequence, if the same goal "branching (equal (parm(y), const(0)), parm(x), ELSE)" is given and the coder knows the schema C7, then the interaction shown in Figure 11 will be produced. Again, at time t_0 the planner delivers the goal. This time the coder instantiates C7 to C7' and implements the ABSYNT tree contained in C7' without requiring subgoals and linking instructions from the planner.

If we compare the first interaction (Figure 10) where the coder knows {O1, O5, L1, L2} with the second one (Figure 11) where the coder knows C7, we observe:

- In the first sequence the coder implements five program fragments corresponding to the subgoals delivered by the planner. In the second sequence the coder implements just one program tree corresponding to the goal.
- In the first sequence the planner gives explicit information about linking program fragments, and the coder rearranges program fragments accordingly, if necessary. In the second sequence there is no such information.

In order to enable *empirical predictions*, we associate the following empirical claims with these observations:

- *Implementation* of ABSYNT program fragments:
  If the coder applies a certain GMR rule, then exactly the ABSYNT program fragment contained in it is implemented in an uninterrupted sequence of programming actions (like positioning a node, drawing a link, etc.). We do not postulate order constrains *within* this sequence, but we expect the sequence not to be interrupted by programming actions stemming from *different* rule instantiations.
- *Verbalization* of goals:
  Following the theoretically motivated distinction of a planner and a coder, selecting goals and subgoals for implementation by the coder is an act of planning involving control decisions. So it seems reasonable that at these decision points the selected goals may be verbalized [17]. The verbalizations explained by the selection of a certain GMR rule may be intermixed with the rule's programming actions, but not with verbalizations and actions stemming from different rule instantiations.

Correction of positions:

If the just implemented program fragment solves a dangling call or calls for another fragment already implemented, then it is to be connected with this existing fragment. Now corrective programming actions are likely: lengthening links, changing their orientation, and moving nodes.

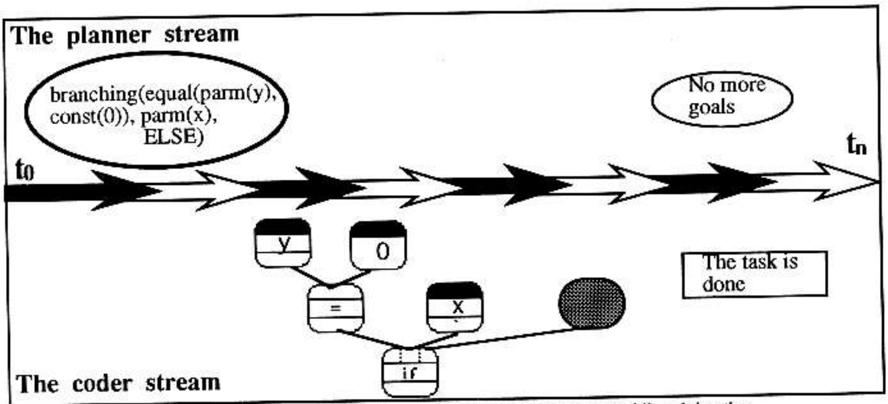


Figure 11: Sequence of interactions between planner and coder while solving the goal "branching (equal (parm(y), const(0)), parm(x), ELSE)" with the schema C7

If we compare the application of a single composite to the application of a set of simple rules (like C7 vs. {O1, O5, L1, L2}), then the following empirical consequences are assumed to result:

- *Implementation* of ABSYNT program fragments (*no-interleaving hypothesis*):
  For the set of simple rules, the order of rule applications is indeterminate, but the programming actions described by each rule should be continuous. *Actions of different rule instantiations should not interleave.* In contrast, when applying the composite there are no order constraints on the programming actions at all since just one rule is applied.
- *Verbalization* of goals (*verbalization hypothesis*):
  In the example, if the coder's knowledge contains C7 the planner has to make one control decision. If the coder knows only {O1, O5, L1, L2}, the planner has to make at least five control decisions (depending on how the goal is decomposed). Thus, we expect that applying composites is accompanied by *fewer goal verbalizations* than applying corresponding sets of simple rules.

- *Correction* of positions (*rearrangement hypothesis)*:
  In case of the composite there are no open GMR calls to be implemented, and there are no to-be-linked program fragments left by earlier rule applications. Thus, we expect that applying composites leads to *fewer position corrections* of ABSYNT nodes and links than applying the corresponding sets of simple rules.
- Performance *time* (*time hypothesis)*:
  Planning, selecting, and verbalizing goals, and correcting positions of nodes and links are internal or external actions that are expected to need time [46]. Thus, we expect that applying composites is *faster* than applying the corresponding sets of simple rules.

These relationships are illustrated in Figure 12 (suppressing the location information for composites) for the rule set {O1, O5, L1, L2}, the composite C7 which may be generated from it, and different sets in between, containing composites and simple rules. The rule sets are organized in a partial order which reflects the *degree of predictability of the order* of programming actions, the *degree of verbalization, position corrections,* and *performance time.*

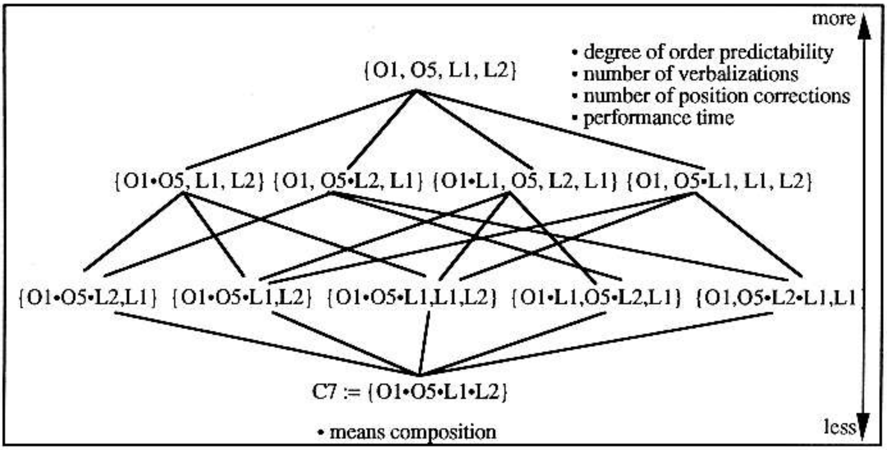


Figure 12: Rule sets partially ordered according to expected degree of order predictability, number of verbalizations, position corrections, and performance time

For example, if the rule set {O1, O5, L1, L2} is applied to the goal "branching (equal (parm(y), const(0)), parm(x), ELSE)", the planner has to chop this goal tree because the coder's knowledge contained in the set {O1, O5, L1, L2} is not sufficient to implement this highly structured goal. If the goal tree is chopped to the stream of goals and goal-subgoal-relations

(branching(IF, THEN, ELSE),
equal(S1, S2),
goal_subgoal_relation(branching(IF, THEN, ELSE), IF, equal(S1, S2)),
parm(y),
goal_subgoal_relation(equal(S1, S2), S1, parm(y)),
parm(x),
goal_subgoal_relation(branching(IF, THEN, ELSE), THEN, parm(x)),
const(0),
goal_subgoal_relation(equal(S1, S2), S2, const(0))),

then the stream of *event sets* (event-set(O1') < event-set(O5') < event-set(connect(O1', 1, O5')) < event-set(L1') < event-set(connect(O5', 1, L1')) < event-set(L1'') < event-set(connect(O1', 2, L1'')) < event-set(L2') < event-set(connect(O5', 2, L2'))) should be observed empirically, where:

- A < B means that the events in event-set A are followed by the events in set B
- event-set(O1') = {verb(branching(IF, THEN, ELSE), act(if-then-else), act(link(if-then-else, 1)), act(link(if-then-else, 2)), act(link(if-then-else, 3))}
- event-set(O5') = {verb(equal(S1,S2)),act(=),act(link(=,1)), act(link(=,2))}
- event-set(connect(O1',1,O5')) = {verb(connect(branching(IF,THEN,ELSE), IF, equal(S1,S2))),act(connect(link(if-then-else), 1, =))}
- event-set(L1') = {verb(parm(y)), act(parameter-node(y)), act(parameter-name(y))}
- event-set(connect(O5',1,L1')) = {verb(connect(equal(S1, S2), S1, parm(y))), act(connect(link(=), 1, parameter(y)))}
- event-set(L1'') = {verb(parm(x)), act(parameter-node(x)), act(parameter-name(x))}
- event-set(connect(O1',2,L1'')) = {verb(connect(branching(IF,THEN,ELSE),THEN,parm(x))), act(connect(link(if-then-else),2, parameter(x)))}
- event-set(L2') ={verb(const(0)),act(constant-node(0)),act(constant value(0))}
- event-set(connect(O5',2, L2')) ={verb(connect(equal(S1, S2), S2, const(0))), act(connect(link(=), 2, constant(0)))}

The empirical meaning of the terms is:

- verb(Goal): The Goal is possibly verbalized.
- verb(connect(Goal1,S,Goal2)): It is possibly verbalized that the subgoal S of Goal1 is Goal2.

- act(Node): The Node is necessarily implemented in ABSYNT in a free region or on a link shadow.
- act(link(Node, I)): An ABSYNT link entering the I-th input of Node is necessarily implemented. Its other end is connected to another node or left as a shadow to be filled later.
- act(connect(link(N1),I,N2)): The ABSYNT link entering the I-th input of node N1 is connected to node N2. (That is, N2 is dragged onto the shadow at the upper end of the link, and/or the link is lengthened to N2.)

The planner may deliver the stream of goals and goal-subgoal-relations in a different order, like the one depicted in Figure 10. Then the order of the empirical event sets should change accordingly. But in any case, the actions and verbalizations *within* each event set should occur in an *uninterrupted sequence*. In contrast, there is no order predictability for the actions and verbalizations corresponding to the *schema* C7, and there is no information about goal-subgoal-relations. Just one set of events can be predicted:

- event-set(C7') = {verb(branching(IF, THEN, ELSE), verb(equal(S1, S2)), verb(parm(x)), verb(parm(y)), verb(const(0)), act(if-then else), act(=), act(parameter-node(y)), act(parameter name(y)), act(parameter-node(x)), act(parameter-name(x)), act(constant-node(0)), act(constant-value(0)), act(link(if-then-else, 1)), act(link(if-then-else, 2)), act(link(if-then-else, 3)), act(link(=, 1)), act(link(=, 2))}

We started to investigate some of these predictions empirically (see below). In addition, the no-interleaving hypothesis and the time hypothesis are used in the construction of the Internal Model to be described now.

7 The Internal Model (IM)

The IM is a set of domain specific knowledge fragments (simple GMR rules and composites) which are utilized and continuously updated. As stated earlier, the IM covers the subset of the ISP-DL Theory shaded in Figures 1 to 4. So before describing it in detail, we will sketch it in terms of the ISP-DL Theory.

- *Concerning Figure 1*: The PS is faced with a programming task (*goal*) and constructs a solution proposal (*solution*). The solution is parsed, using the *knowledge base* (rules in the IM and - as far as needed - in EXPERT). Subsequently, the rules just used for parsing are *optimized* by composition.

Since these new composites may be based on EXPERT rules, they are not directly inserted into the IM: according to ISP-DL Theory, a rule can only be improved after its successful application. This applies to the IM in that it cannot at the same time be augmented by a new simple rule (from EXPERT) and by composites built from the same simple rule. For this reason, in addition to the IM there is a set POSS of possible candidates for future composites of the IM. Composites of the rules used for parsing a solution proposal are generated and kept in POSS as candidates. Only those surviving a later test are moved into the IM.

- *Concerning Figure 2*: If parsing the solution is possible solely with rules in the IM, then the IM is considered as sufficient to construct the solution, and "Goal Processing" is terminated ("*reaction to success*"). But if parsing the solution requires additional EXPERT rules, then the IM may be augmented by these (simple) rules ("*inductive knowledge acquisition*").

  Thus, in accordance with ISP-DL-Theory, the IM contains *simple rules* representing newly acquired but not yet improved knowledge, and *composites* representing various degrees of expertise.

- *Concerning Figure 3*: The parse tree represents the student's hypothetical solution *plan*, whose *execution* led to a *protocol*: the sequence of programming actions, verbalizations, and corrections exhibited by the student. We call that part of the protocol consisting only of the student's programming actions (creating nodes and links, naming nodes) the student's *action sequence*. The action sequence is used to evaluate the parse rules:
  - Since knowledge improvement should result in sped-up performance (*time hypothesis*), a composite is moved from POSS to IM only if the PS shows a *speedup from an earlier to a later action sequence* where both sequences can be produced by the composite.
  - The IM contains only GMR rules (simple rules and composites) which proved to be *plausible* with respect to an action sequence at least once. This is defined now. With respect to some action sequence, GMR rules form four subsets:
    1. Rules not containing any program fragments ("goal elaboration rules") are *nondecisive* with respect to the action sequence. (But verbalizations can be related to the goal elaboration rules [42]).
    2. Rules whose head contains a program fragment which is part of the final result produced by the action sequence, and which was programmed in a

noninterrupted, temporally continuous subsequence (see the *no-interleaving hypothesis*). These rules are *plausible* with respect to the action sequence.

3. Rules also containing a program fragment which is part of the final result of the action sequence, but this fragment corresponds only to the result of a *non*continuous action subsequence *interrupted* by other action steps. These rules are *implausible* with respect to the action sequence.
4. Rules whose head contains a program fragment which is not part of the final result produced by the action sequence. These rules are *irrelevant* to the action sequence.

- A *credit* scheme rewards the usefulness of the rules in the IM. The credit of a rule is the total number of action steps explained by this rule in the problem solving process of the PS. It is the product of the length of the action sequence explained by the rule and the number of its successful applications. Thus the credit depends on the empirical evidence gathered for a rule.

During the knowledge acquisition process the IM is utilized and continuously updated according to a processing cycle shown in Figure 13:

- *Start* (Top of Figure 13): The first programming task is presented. Initially, both sets IM and POSS are empty.
- Now the learner solves the first task presented. Thus an *action sequence* is produced, leading to a *solution* to the task. The action sequence is saved in a log file.
- *First Test:* IM and POSS are empty, so nothing happens.
- *First Parse:* The learner's ABSYNT program solution to the actual task is parsed with the EXPERT rules, leading to a set of parse rules.
- *First Generate*: The EXPERT rules just used for parsing are compared to the action sequence. The *plausible* parse EXPERT rules are put into the IM and get credit. These rules are hypothesized as newly acquired and applied by the PS while solving the first task.

  Next, the composites of all parse rules are created and compared to the action sequence. The plausible composites are kept in POSS. These rules are hypothesized as newly created as a result of success-driven learning, but not yet actually used. Thus they are candidates of improved knowledge useful for future tasks. To each plausible composite, the time needed by the PS to perform the corresponding action sequence is attached.

  So the Generate phase results in an updated POSS and IM.
- Now the next task is presented to the PS. The PS creates an ABSYNT action sequence and solution to it.

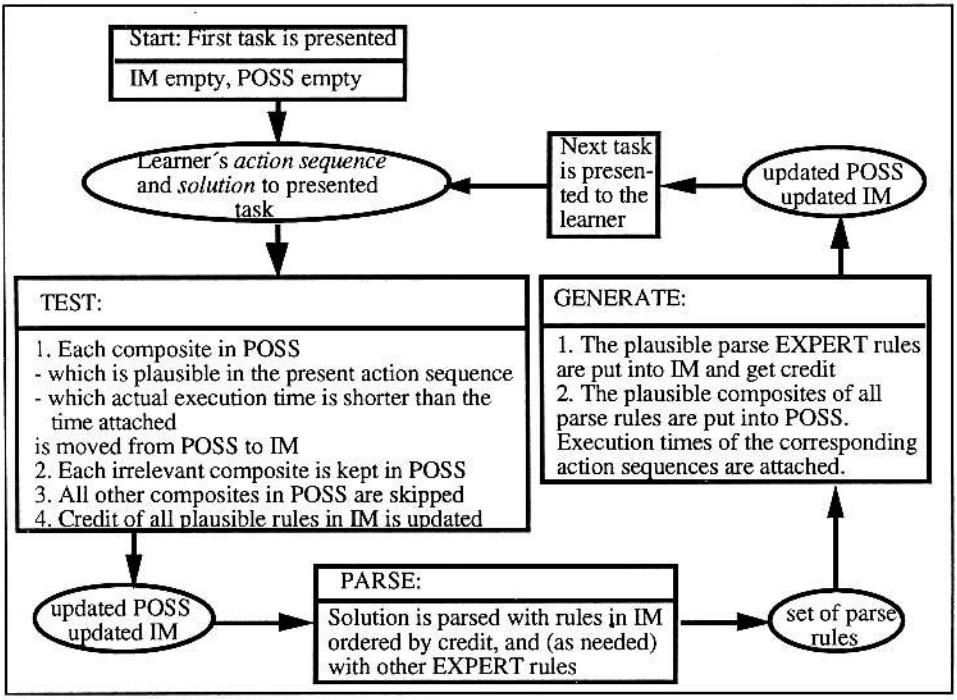


Figure 13: The utilizing and updating cycle of the IM during the knowledge acquisition process

- *Second Test*: Each composite in POSS is checked if
  a) it is plausible with respect to the action sequence, and
  b) the time needed by the PS to perform the respective continuous action sequence is shorter than the time attached to the composite. This means that the PS performs the action set *faster* than the previous corresponding action set which was shown by the PS before the hypothesized creation of the composite.

  The composites meeting these requirements are put into the IM. Composites irrelevant to the action sequence of the solution just created are left in POSS. They might prove as useful composites on future tasks. All other composites violate the two requirements. They are skipped: that is, composites implausible to the actual sequence, or composites which predict a more speedy action sequence than observed. This means that the PS performs the action set *slower* than the previous corresponding action set which led to the creation of the composite. This slow-down is inconsistent with our model assumption that the PS prefers composites to simple rules; thus the composite is not transferred to the IM but skipped. Finally, the credits of all rules in the IM which are plausible with respect to the present action sequence are updated. Thus the second test leads to an updated POSS and IM.

- *Second Parse*: Now the solution of the second task is parsed with the rules of the IM ordered by their credits. As far as needed, EXPERT rules are also used for parsing.
- *Second Generate*: The plausibility of EXPERT rules which have just been used for parsing is checked. The plausible EXPERT parse rules are again put into the IM and get credit. As in the first Generate Phase, they are hypothesized as the newly acquired knowledge in response to impasses on the task just performed. Furthermore, the composites of all actual parse rules are created. The plausible composites are put into POSS, they will be tested on the next test phase. Again the time needed for the corresponding action sequence is stored with each composite.

8 Illustrations of the IM

To illustrate, Figure 14 shows a continuous fragment of the action sequence of a PS, Subject 2 (S2), on a programming task. Again we will restrict our attention to the rules O1, O5, L1, L2, and C7 (see Figures 8 and 9). When S2 performs the sequence of Figure 14, O1, L1 and L2 are already in the IM from earlier tasks. O5 is not yet in the IM but only in the set of EXPERT rules. C7 has not yet been created.

After S2 has solved the task, the *Test Phase* (Figure 13) starts. Since the only composite we look at here (C7) has not been created, we only consider the fourth subphase: Credit updating. O1 is *implausible* with respect to Figure 14 because the actions corresponding to the rule head of O1 are not continuous but *interrupted*. They are performed at 11:15:52, 11:15:58, 11:16:46, and 11:16:55 (Figure 14). Thus the action sequence corresponding to the rule head of O1 is interrupted at 11:16:42 and 11:16:50.

L1 and L2 are also implausible. Actions corresponding to L1 are performed the first time at 11:15:08 and 11:15:29. Thus this sequence is interrupted at 11:15:16 and 11:15:22. L1-like actions are shown a second time by the PS at 11:16:42 and 11:16:50. These are interrupted, too. Actions corresponding to L2 are performed at 11:15:16 and 11:15:34, with interruptions at 11:15:22 and 11:15:29. So since O1, L1, and L2 are implausible, their credits are not changed.

Now S2's solution is *parsed* with rules in the IM and, as needed, with additional EXPERT rules (Figure 13). O1, O5, L1, and L2 are among the parse rules in this case, as no other rules have a higher credit and are able to parse the solution.

After the Parse Phase, the *Generate Phase* (Figure 13) starts. O5 is an EXPERT rule used for parsing. But O5 is implausible, since its corresponding actions were performed at 11:15:22,

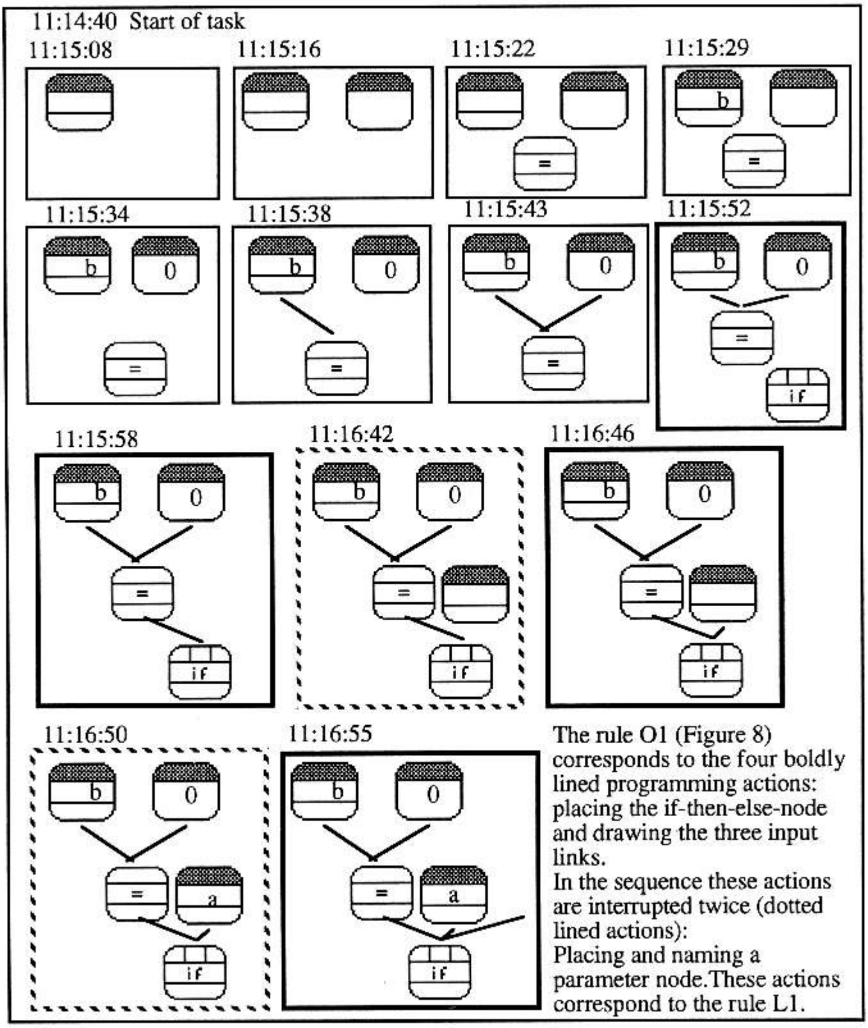


Figure 14: A continuous fragment of a sequence of programming actions of Subject S2

11:15:38, and 11:15:43, with interruptions at 11:15:29 and 11:15:34. So O5 is not put into the IM. Then the composites of the parse rules are formed. C7 (Figure 9) is a composite formed from O1, O5, L1 and L2. This composite is plausible because it describes the uninterrupted

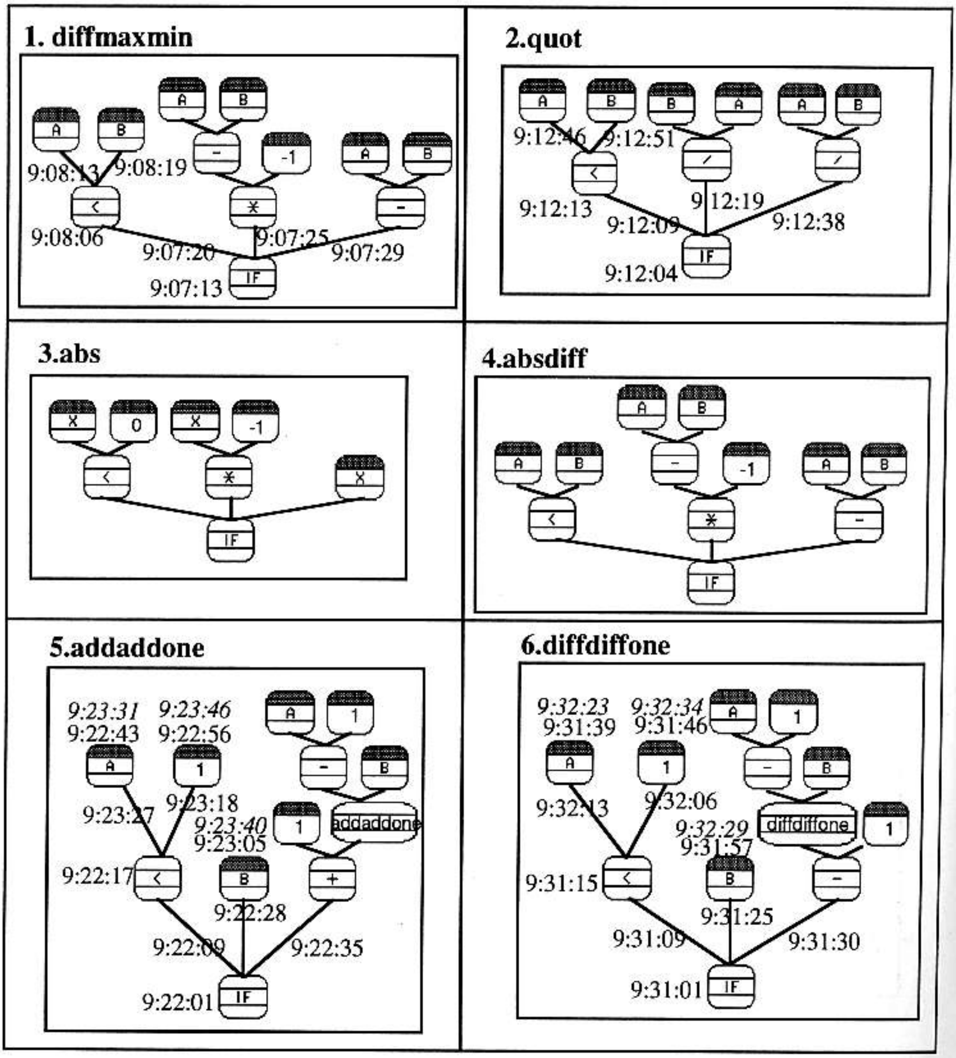


Figure 15: Simulated action streams and solution proposals to 6 ABSYNT programming tasks

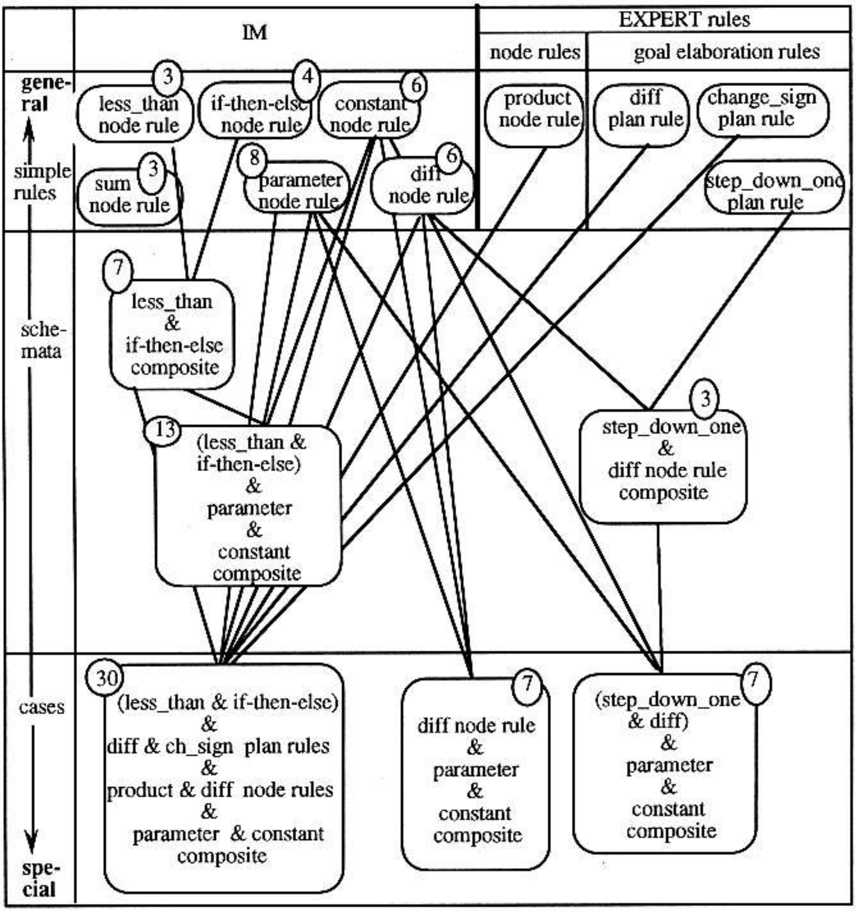


Figure 16: Specialization graph showing the partial order of simple rules, schemata, and cases as a result of the simulated programming sequence of Figure 15

sequence of programming actions from 11:15:08 to 11:16:55 (see Figure 14) - despite the fact that its components O1, O5, L1, and L2 are all implausible. Starting from the beginning of the task (at 11:14:40), the time for this action sequence is 135 seconds. Thus the composite C7 is stored in POSS with "135 seconds" attached to it.

After S2 has solved the next task, the following Test phase reveals that C7 is plausible again. The corresponding action sequence (not depicted) was performed in 92 seconds, which is less than 135. So C7 is moved into the IM and gets a credit of 13 since it describes 13 programming steps (see Figure 14). This credit will be incremented by 13 each time the composite is plausible again.

What does the IM look like after several tasks are solved? To give an impression we simulated the protocol of a virtual PS, starting with an empty IM and POSS. Figure 15 depicts the body trees of the solutions of 6 ABSYNT programming tasks: "diffmaxmin" (subtraction of the smaller from the larger of two numbers), "quot" (division of the larger by the smaller of two numbers), "abs" (absolute value of a number), "absdiff" (like "diffmaxmin": absolute difference of two numbers), "addaddone" (expressing addition by "+ 1"), "diffdiffone" (expressing subtraction by "- 1"). Some of the programming actions leading to these solutions are labeled with the time when they were performed. For example, the "<"-node in the solution to the task "diffmaxmin" was programmed at 9:08:06. The link between the "<"-node and the "if-then-else"-node was created at 9:07:20.The times of the actions of writing a value or name into a node are written in *italics*.

After solving the last task of this sequence, "diffdiffone", the IM contains simple rules, schemata, and cases. They can be ordered as a specialization graph, as shown in Figure 16. (The rules in Figure 16 are depicted in PROLOG in Appendix A.) The circled numbers are the credits. Each composite in Figure 16 is connected to the rules it is built from. For example, the "(less_than & if-then-else) & parameter & constant" composite in Figure 16 is:

gmr(branching (less_than (parm (Y), const (C)), parm (X), S),
    if-pop (lt-pop (Y-pl, C-cl), X-pl, P)) :-
is_parm (Y), is_const (C), is_parm (X), gmr (S, P).

("lt-pop" is the primitive ABSYNT operator "<".)

According to Figure 16, this composite is the result of composing the "less_than & if-then-else" composite with the parameter node rule L1 and the constant node rule L2 presented earlier:

"less_than & if-then-else" composite:

gmr(branching(less_than(S1,S2), S3, S4), if-pop(lt-pop(P1, P2), P3, P4)) :-
    gmr(S1, P1), gmr(S2, P2), gmr(S3, P3), gmr(S4, P4).

L1: gmr(parm(P), P-pl):- is_parm(P).

L2: gmr(const(C), C-cl):- is_const(C).

This composition can be expressed by

"(less_than & if-then-else) & parameter & constant" composite =

$((\text{"less\_than \& if-then-else" composite}_1 \bullet \text{L1})_1 \bullet \text{L2})_1 \bullet \text{L1}_1$.

A few examples will demonstrate how the IM in Figure 16 develops for the simulated programming sequence of Figure 15:

- Initially the IM is empty, so the solution to the first task (diffmaxmin) is parsed with EXPERT simple rules. The if-then-else-node rule (rule O1 shown earlier) and the less_than node rule are among the parse rules of the solution of the first task. The times attached to the solution in Figure 15 show that these rules are plausible. For example, the "if-then-else"-node and the three links leaving it were programmed in a continuous uninterrupted sequence (four programming actions from 9:07:13 to 9:07:29). The same is true for the "<"-node and the two links leaving it (three programming actions from 9:08:06 to 9:08:19). So these two rules get into the IM and get the credits 4 and 3, respectively.
- Among the composites built from the parse rules of the solution to diffmaxmin, there is a schema, the "less_than & if-then-else composite". It is also plausible so it is moved into POSS. The action sequence explained by this composite starts at 9:07:13 and ends at 9:08:19, so the time "66 seconds" is attached to it.
- After solving "quot", the "less_than & if-then-else composite" is plausible again. Additionally, the corresponding action sequence is faster than 66 seconds (from 9:12:04 to 9:12:51, which is 47 seconds). So this composite is moved into the IM and gets a credit of 7 since it describes 7 programming actions.
- Another example is the "(less_than & if-then-else) & parameter & constant composite". The corresponding 13 actions are performed at the task "addaddone" in a continuous sequence (from 9:22:01 to 9:23:46, which is 105 seconds). Thus this schema is plausible and is put into POSS. On the next task, diffdiffone, this composite is plausible again, and the corresponding action sequence is sped up (from 9:31:01 to 9:32:34, which is 93 seconds). So the schema gets part of the IM with a credit of 13.

Figure 16 also shows that composites may be in the IM but not the simple rules they originate from. For example, the product node rule is not part of the IM but has been used for creating a case which is in the IM.

9 An Empirical Analysis of the IM

The IM represents the actual hypothetical knowledge of the PS. In this section we will investigate the no-interleaving hypothesis stating that the programming actions described by a rule in the IM are performed in a continuous uninterrupted temporal sequence. We will also take a look at some verbalizations, position corrections, and performance times. The analysis is based on the programming actions performed by a single subject, S2, solving seven consecutive nonrecursive ABSYNT programming tasks. The IM was run offline based on the action sequences exhibited by S2, because we had videotaped and categorized the session before.

- *Material and procedure.* In a "getting-started" phase, S2 constructed an ABSYNT Start tree for each primitive ABSYNT operator node, and reconstructed given programs. The purpose of this phase was to introduce S2 to the ABSYNT interface and language. Then she solved the following tasks: "diffmaxmin", "interval" (program that tests if a number lies between 1 and 2), "absdiff", "quot", "quotzero" (like quot, but preventing division by zero), "abs", and "volume" (program that computes the difference between the volume of a cube and a sphere, where the diameter of the sphere is equal to the length of the edge of the cube).
- *Creating subsequent states of the IM.* Subsequent states of the IM were created by generating an initial state of the IM and then running it on S2's solution sequence. We created an initial IM based on the following assumption: Since the subject was introduced to all ABSYNT nodes before she worked on the first programming task, "diffmaxmin", it seemed reasonable to put the primitive operator node rules, the constant node rule, and the parameter node rule into the IM. Then the IM was run on the sequence of solutions from "diffmaxmin" to "volume" constructed by S2. This produced a sequence of seven subsequent states of the IM.
- *Analyzing S2's protocol.* The protocol of S2's solutions to the seven programming tasks (S2's complete *subject trace*) was analyzed according to the following categories of events (actions and verbalizations):
    - placing a node
    - naming a parameter, constant, or higher operator node
    - creating a link
    - deleting a node or a link
    - replacing a node by another node, or changing a parameter name or a constant value
    - correcting the position of a node or a link
    - verbalizing a goal to place, name, or replace a node, or to create a link
    - verbalizing uncertainty ("maybe I should ...") or negations ("I don't know whether ...")

  The actions and verbalizations of S2 while working in the Hypotheses Testing Environment were not included in this analysis because our hypotheses are not aimed at this activity.
- *Postdicting action and verbalization sequences* (event sequences). Based on
    - the state of the IM right before each task, and

- S2's event sequence leading to a solution of this task,

the following postdictions for this event sequence were made:

- *Sets* that contain actions of placing and naming nodes, creating links, and verbalizing respective goals *(model trace)*. Each set corresponds to the application of one IM rule. Thus the model trace is a set of sets where each set contains actions and verbalizations expected to occur in a continuous uninterrupted sequence within S2's subject trace (*no-interleaving hypothesis*). The model trace contains no deletions of ABSYNT fragments (nodes, links) since our hypotheses do not cover deletions.
- *Position corrections.* If the position of a node is corrected, the IM rule explaining the corrected node should *not* explain the nodes connected to this node. Rather, these linked nodes should be explained by different rules. If the position of a link is corrected, the IM rule explaining it should not explain the node at the upper end of this link (*rearrangement hypothesis*).
- *Performance times.* An event sequence explained by a composite should be shorter than the earlier event sequence which led to the creation of the composite (*time hypothesis*).
- *Evaluation of the subject trace with respect to the model trace.* For two consecutive tasks of the task sequence, "absdiff" and "quot", Figures 17 and 18 show the actual state of the IM, S2's solution, the subject trace with correspondences (+) and contradictions (-) between model trace and subject trace, and the model trace. The assignment of "+" and "-" will be explained in a moment. More specifically, Figure 17 shows:
  a) a subset of the rules in the IM after solving the task "interval" (second task of the sequence). Here only the rule names are given. The actual rules are shown in Appendix B.
  b) S2's solution to "absdiff", which is the third task in the sequence.
  c) S2's subject trace of the solution to "absdiff" with correspondences (+) and contradictions (-) to the model trace.
  d) The postdicted model trace, given the subject trace and the state of the IM. (Dots in some of the model trace sets stand for actions expected according to the respective IM rule, but not occurring in the subject trace.)

  Figure 18 shows the same information for the next task, "quot". In Figures 17 and 18, the nodes are indexed if necessary in order to avoid ambiguities.

S2's subject trace was compared to the model trace in two equivalent ways. The first method better illustrates which events belong together according to the model trace. The first method leads to the "+" and "-" assignments on the left of the subject traces

(Figures 17c and 18c). The second method better illustrates the relations between adjacent subject trace events. It leads to the assignments on the right of Figures 17c and 18c.

First method:

- If *all* events within one set of the model trace occurred in an uninterrupted temporal sequence in the subject trace, then a "+" was assigned for *each* adjacent pair of this sequence. (Thus the IM rule corresponding to that model trace set is plausible.) For example, lines 5 to 7 of Figure 17c correspond to the third set of Figure 17d.
- If there was at least *one* interruption of this sequence by some other action or verbalization not in the respective model trace set, then a "-" was assigned for *each* pair of this sequence. (Thus the IM rule corresponding to that model trace set is not plausible.) For example, lines 1 and 3 of Figure 17c correspond to the first set of Figure 17d but they are interrupted by line 2. As another example, lines 18, 20, 21, 28, and 41 of Figure 17c correspond to the sixth model trace set of Figure 17d. These lines are not continuous, so *each* pair of them gets a "-".

"+" denote correspondences to the no-interleaving hypothesis, and "-" denote contradictions. This criterion is strong since a single interruption of an otherwise uninterrupted sequence causes *all* pairs of events of this sequence to be counted as "-". (For example, lines 20 and 21 of Figure 17c get a "-" although they are continuous. As another example, the pair of lines 8 and 9 of Figure 18c stem from the same model trace set, but this pair gets a "-" since the other events of this model trace set do not occur continuously, but at lines 18 and 24.) This is required by the no-interleaving hypothesis and the plausibility criterion of the IM.

a) Subset of the rules in the IM before S2 solves the task "absdiff (after solving "interval")

if-then-else node rule	less_than node rule(lt)	parameter node rule
difference node rule	less_than node rule(gt)	constant node rule
product node rule	less_equal node rule	

b) S2's solution to the task "absdiff" (program body)

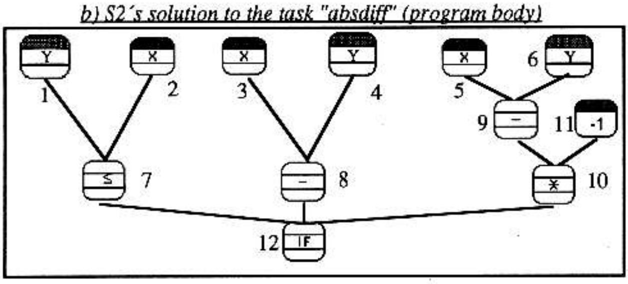


Figure 17 (continued on the next page): a) subset of the IM before solving "absdiff", b) S2's solution to "absdiff", c) subject trace with correspondences (+) and contradictions (-), and d) model trace

		c) subject trace		*d) model trace*
1		place node1 parameter	-	{place node1 parameter, name node1 X}
2	-	place node2 parameter	-	{place node2 parameter, name node2 Y}
3		name node1 X		
4	-	name node2 Y		
5		place node7 -		{place node7 -,
6	+	create link from node7 -	+	create link from node7 -
		to node1 X	+	to node1 X,
7	+	create link from node7 -		create link from node7 -
		to node2 Y		to node2 Y}
8		verbalize uncertainty: > or < or		{verbalize uncertainty: > or < or
		if-then-else for node7 - ?	-	if-then-else for node7 - ?,
9	-	replace node7: - by ≤		
10		place node12 if-then-else		place node12 if-then-else,
11	-	create link from node12	-	create link from node12
		if-then-else to node7 -		if-then-else to node7 -, ..., ...}
12		replace node7: ≤ by <		
13		delete node12 if-then-else		
14		delete node7 <		
15		delete link from node12 to node7		
16		place node7 -		
17		delete node7 -		
18		verbalize goals: if-then-else, >	+	{verbalize goals: if-then-else, >,
19	- +	place node7 >	-	place node7 >, ..., ...}
20		place node12 if-then-else		{verb. goal if-th.-el., pl. node12 if-th.-el.,
21	-	create link from node12	-	create link from node12
		if-then-else to node7 >	-	if-then-else to node7 >,
22	-	place node3 constant		create link from node12 to node8 -,
				create link from node12 to node10 *}
23	-	replace node3: constant		{replace node3: parameter for constant,
	-	by parameter	-	name node3 X}
24		place node4 parameter	-	{place node4 parameter, name node4 Y}
25	+	place node8 -		{place node8 -,
26	+	create link from node8 - to node3	+	create link from node8 - to node3,
27		create link from node8 - to node4	+	create link from node8 - to node4}
28		create link from node12 if-then-		
	-	else to node8 -	-	
29		name node3 X		
30		name node4 Y		
31	-	place node5 parameter	-	{place node5 parameter, name node5 X}
32		place node6 parameter	-	{place node6 parameter, name node6 Y}
33		place operator9 -		{place operator9 -,
34	+	create link from node9 - to node5	+	create link from node9 - to node5,
35	+	create link from node9 - to node6	+	create link from node9 - to node6}
36	-	place node11 constant	-	{place node11 constant, name node11 -1}
37		place node10 *	+	{place node10 *,
38	+	create link from node10 * to node9 -	+	create link from node10 * to node9 -,
39	+	create link from node10 *		create link from node10 *
		to node11 constant		to node11 constant}
40		name node11 -1		
41		create link from node12 to node10 *		
42		name node5 X		
43		name node6 Y		
44		verbalize uncertainty: switch		{verbalize uncertainty: switch
	+	names of node1 and node2?	+	names of node1 and node2?,
45	-	replace node1: X by Y		replace node1: Y for X}
46		replace node2: Y by X	-	{verb. uncert., replace node2: X for Y}
47		replace node7: > by ≤		

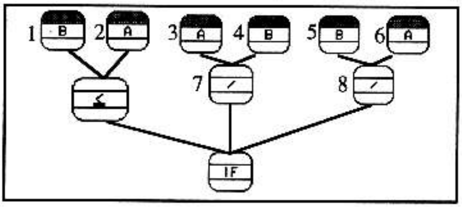
a) Subset of the rules in the IM before S2 solves the task "quot" (after solving "absdiff")

if-then-else node rule
less_equal node rule

quotient node rule
sum node rule
difference node rule

parameter node rule
constant node rule

b) S2's solution to the task "quot" (program body)



	c) subject trace	*d) model trace*
1	place node1 parameter	{place node1 parameter, name node1 b}
2	place node2 parameter	{place node2 parameter, name node2 a}
3	place operator ≤	{place operator ≤,
4	create link from ≤ to node1	create link from ≤ to node1,
5	create link from ≤ to node2	create link from ≤ to node2}
6	name node1 b	
7	name node2 a	
8	place op. if-then-else	{place op. if-then-else,
9	create link from if-then-else to ≤	create link from if-then-else to ≤, create link from if-th.-else to node7 /, create link from if-th.-else to node8 /}
10	place node3 parameter	{place node3 parameter, name node3 a}
11	place node4 parameter	{place node4 parameter, name node4 b}
12	place node7 /	{place node7 /,
13	create link from node7 to node3	create link from node7 to node3
14	create link from node7 to node4	create link from node7 to node4}
15	name node3 a	
16	name node4 b	
17	correct position of if-then-else and link from it-then-else to ≤	{correct position of if-then-else and link from it-then-else to ≤}
18	create link from if-then-else to node7 /	
19	place node5 parameter	{place node5 parameter, name node5 b}
20	place node6 parameter	{place node6 parameter, name node5 a}
21	place node8 /	{place node8 /,
22	create link from node8 to node5	create link from node8 to node5
23	create link from node8 to node6	create link from node8 to node6}
24	create link from if-then-else node to node8 /	
25	name node5 b	
26	name node6 a	

Figure 18: a) subset of the IM before solving "quot", b) S2's solution to "quot", c) subject trace with correspondencies (+) and contradictions (-), and d) model trace

Instead of assigning "+" or "-" to *pairs* of events contained in the same model trace set, a "+" or "-" could be assigned only to *complete* sequences of the subject trace which correspond to a model trace set. But this does not account for the number of action steps explained by the supposed IM rule. (The parameter node rule explains two action steps, but composites can explain 30 or more.)

Second method: For each *adjacent* pair of events of the subject trace:

- If both events are contained in a model trace set which events occur in an uninterrupted sequence in the subject trace, then a "+" is assigned. For example, the pair of events in lines 3 and 4 of Figure 18c gets a "+", since all events of the corresponding model trace set (the third set of Figure 18d) are continuous in the subject trace.
- If both events are contained in the same model trace set but the events of this set do not occur in an uninterrupted sequence in the subject trace, then a "-" is assigned. For example, in Figure 18c a "-" is assigned to the pair of lines 8 and 9, because the events of the respective model trace set {place operator if-then-else, create link from if-then-else to ≤, create link from if-then-else to node7 /, create link from if-then-else to node8 /} do not form an uninterrupted sequence in the subject trace.
- If the two events are not contained in the same model trace set, and the first of these two events is contained in a model trace set having at least one more event later in the subject trace, then a "-" is assigned. For example, in Figure 18c a "-" is assigned to the pair of events in lines 1 and 2. The first action ("place node1 parameter") belongs to the model trace set {place node1 parameter, name node1 b} which contains another action ("name node1 b") occurring later in the subject trace.
- If the two events are not contained in the same model trace set, and the first one of these two events is the finishing event of the model trace set it belongs to (the last event of this set in the subject trace), then nothing is assigned. For example in Figure 18c nothing is assigned to the pair of lines 5 and 6, since the event of line 5 "create link from ≤ to node2" is in a model trace set whose other actions occurred earlier in the subject trace.
- *Results*
    - *Comparison of model trace and subject trace*. For S2's complete subject trace (for all seven tasks), there were 76 "+" and 60 "-" indicators. Since more "+" indicators should lead to longer and thus fewer runs (continuous sequences of "+" or "-", for example, the sequence "++---+" has three runs) than an equal distribution of "+" and "-", we applied the Runs-test to the sequence of "+" and "-" as obtained by the second method (on the right of Figures 17c and 18c). There were 42 runs in S2's complete subject trace, significantly less than to be expected by chance ($p < 0.001$). This confirms our no-interleaving hypothesis.

- *Position corrections.* S2's complete subject trace contained six position corrections. One of them occurs in the subject trace of "quot" (Figure 18c). There were three node corrections of parameters and constants. They were explained by different rules than the nodes connected to them. There were also three corrections of operator nodes and one of their input links. (In Figure 18c, the "if-then-else" node and its first input link are rearranged.) They were also explained by different rules than the node at the upper end of the respective link. (In Figure 18c, the "if-then-else" node and the "≤" node are explained by different IM rules.) So all position corrections are consistent with the rearrangement hypothesis.
- *Performance time.* Only one action sequence of S2's complete subject trace is explained by a composite. It shows a speedup (from 387 to 211 seconds) which is consistent with the time hypothesis.

- *Discussion*

The results indicate that the IM adequately describes more than half of the protocol of S2's actions and verbalizations with respect to the no-interleaving and rearrangement hypotheses. There was only one action sequence relevant to the time hypothesis. We will discuss several points raised by this analysis:

- *Time patterns.* There is another observation about time. The complete subject trace contained 29 event sequences denoted by a series of "+" indicators, thus, corresponding to a set of the model trace. (For example, in Figure 18c, lines 12 to 14 form such a sequence as expected by the set {place node7 /, create link from node7 to node3, create link from node7 to node4} of the model trace). For 23 of these 29 action sequences, their first action takes more time than each of the other actions. This is exactly what we would expect since according to our model, before the *first* action of an IM rule is executed, the planner has to generate a goal, and the coder has to look for and to select that rule.
- *Discrepancies.* There is of course a large amount of 60 discrepancies ("-"). How can they be explained?

  One possibility is that our criterion for assigning "+" and "-" is too strong because it does not allow for "partial" evidence for an IM rule. For a sequence of the subject trace corresponding to a model trace set, each pair of the sequence gets a "-" even if there is only one interruption. So an alternative is to weaken the no-interleaving hypothesis (and the plausibility concept used for IM creation as well) in the following way: a "+" is assigned to each pair of events which is contained within the same set of the model trace, and which is adjacent in the subject trace. (For example, in Figure 18c the pair of lines 8 and 9 "place operator if-then-else" and "create link from if-then-else to ≤" now gets a "+".) Correspondingly, a "-" is assigned to each pair within the same model trace set but interrupted by some other action(s). Using this criterion, there were 84 "+" and 52 "-" indicators in S2's complete subject trace. But this still leaves a lot of "-" indicators left to be explained.

Another observation with respect to the no-interleaving hypothesis is that a large portion of the discrepancies seems to be caused by parameters and constants. Table 1 shows the distribution of "+" and "-" across different types of rules in the IM:

	Parameter node rule	Constant node rule	Primitive operator node rules	Composites
"+" cases	3	4	46	23
" -" cases	28	8	24	0

Table 1: Distribution of "+" and "-" across different types of rules in the IM after solving the sequence of seven programming tasks

Thus the parameter node rule, for example, is responsible for 3 "+" and for 28 "-" indicators: S2 usually does not place and name a parameter node in sequence. The same seems true for the constant node rule. Given that this result will be reproduced with traces of other subjects, it seems that the no-interleaving hypothesis cannot be maintained for parameters and constants. There are two ways to cope with this:

1. to *split* the parameter node rule and the constant node rule into two new rules: one for positioning and one for naming a parameter node or constant node, respectively. Then the current parameter and constant node rule would be considered as a *composite* of more primitive rules explaining only one programming action.
2. to allow rule applications to be *interrupted.* Perhaps once S2 had acquired a rule, she was more flexible in applying it than stated by the no-interleaving hypothesis. This would mean that IM rule applications can be temporarily interrupted by the application of other rules. So if an IM rule is applied, some of the events of the corresponding model trace set might not be adjacent in the subject trace.

But this interruption hypothesis needs to be constrained further. Firstly, we would propose that interruptions of rule applications should not consist of deletions and replacements, since these actions indicate replanning which should cause the interrupted rule application to stop. Secondly, in terms of the planner-coder interaction, to interrupt a rule application means that the planner switches to another goal and the coder selects a rule to implement that goal. This should take additional time. So the times between adjacent events in the subject trace denoted by "-" should be longer than the times for pairs denoted by "+". These hypotheses will be investigated further when more information about the process of program planning is available by incorporating planning nodes into ABSYNT (see below).

- *Replacements*. Replacements were not considered in this analysis, but they could be handled in the following way. Several replacements occur in the subject trace of Figure 17c. For example, the first seven actions of subject S2 (lines 1 to 7) result in a program tree T1 consisting of two parameters, X and Y, and a "-" node linked to them. But then she feels uncertain (line 8), considers "if-then-else", ">", and "<", and then replaces the "-" node by a "≤" node (line 9). One might think of this replacement as a shortcut for deleting T1 and constructing a new tree T2 consisting of a "≤" node linked to two parameters, X and Y. In addition, we might assume that T2 would be constructed in the same manner as T1 before. So since the construction of T1 led to two "-" and two "+" indicators, two "-" and two "+" indicators should as well be assigned to the replacement action in line 9 ("-" by "≤") since this action is viewed as a shortcut for constructing T2.

  Altogether, S2's subject trace contains 13 replacements, 5 of them occurring at the task "interval" and 8 at "absdiff" (see Figure 17c. There are six immediate replacements, denoted as "replace node... ", and two delayed replacements where deletion of the to-be-replaced node and placing the new node are interrupted by another action. The first one of these two cases occurs in lines 14 and 16 of Figure 17c: "delete node7 <" ... "place node7 -", and the second, similar case consists of lines 17 and 19.) If we account for replacements in the way just described, we obtain 106 "+" and 90 "-" indicators for S2's complete subject trace.
- *Composites*. By the end of the last task ("volume"), there were only two composites in the IM. The virtually created programming sequence shown in the preceding section led to six composites (three schemata, three cases: Figure 16) after solving six tasks, and even more composites would have been possible. Thus according to the IM, subject S2 does not make much use of her own previous solutions but does much problem solving. This conclusion is supported by an inspection of the solutions of S2 to the seven tasks. For example, she solves "diffmaxmin" by "maximum of a and b minus minimum of a and b", but she solves the essentially

identical task "absdiff" by "if b less than a then a minus b else (a minus b) times -1". (As can be expected by the large number of replacements, the "absdiff" task seems to be much harder for S2 than "diffmaxmin", even though "diffmaxmin" was solved earlier.) Subsequently, the task "quot" is solved in yet another way by interchanging parameters. Thus the diversity in solution approaches is reflected in the IM by the fact that it contains only few composites.

- *Impasses*. Based on S2's IM we cannot predict impasses because
  - the IM currently contains *only* implementation knowledge ("the coder's knowledge") but no planning knowledge. (We are working on extending the IM in this way.)
  - the IM contains *sufficient* implementation knowledge because, as stated, it contains all primitive node rules and parameter and constant rule from the beginning.

  So there should be no impasses based on insufficient implementation knowledge. Consequently, all impasses in the protocol should be attributable to insufficient planning knowledge. If we propose verbalizations of uncertainty and negative comments as one empirical criterion for an impasse (similar to [57, 60]), then the protocol contains five impasses (without the hypotheses testing episodes). Four of them occur at the tasks "interval" and "absdiff" which seemed to be most difficult for S2. In three impasse situations S2 considers different implementations ("if-then-else" or a logic operator; ">" or "<", and so on) and is uncertain about them. (An example occurs at line 8 of Figure 17c). Thus there appears to be a planning problem. In the fourth case the impasse arises because S2 thinks that the solution just created will deliver a wrong result for a critical input value. In response to this, S2 switches parameter names. This does not seem to be an implementation problem either.

  We are working on extending the ABSYNT Problem Solving Monitor and the IM by incorporating a planning level (see below). Then it should be possible also to predict impasses based on missing planning knowledge.

10 Discussion

We presented an approach to online diagnosis of students' knowledge states which is aimed at meeting the following requirements:

- to be based on a theoretical problem solving and learning framework,

- to be computationally effective and empirically valid,
- to support adaptive help generation.

We will now discuss how far the IM meets these requirements and how we plan to improve it.

- *Foundation on a theoretical framework.* In section 7 we showed how in our view the IM is related to the ISP-DL Theory. We tried to motivate the features of the IM by the theory. But still many aspects of the theory remain uncovered by the IM. Two of them are:
  - *Generalization* of knowledge. Our observations from single-subject sessions with ABSYNT indicated use of previous solutions and positive transfer especially for recursive tasks. Thus composites in the IM should be generalized. *Generalization of composites* may be viewed as another way of knowledge optimization (eg.[1, 65]) in response to the successful utilization of knowledge (Figure 1). Additionally, generalized knowledge should also result from *analogizing* as an alternative to synthesizing a plan (Figure 3).
  - Synthesizing a *plan*. Currently the IM takes only account of the implementation level, but there is no representation of planning knowledge within the IM.

We will sketch our current work on these two aspects:

- Concerning *generalization*, we will consider a simple example. We suppose that:
  a) The two fragments shown in Figure 19 were programmed on two consecutive tasks

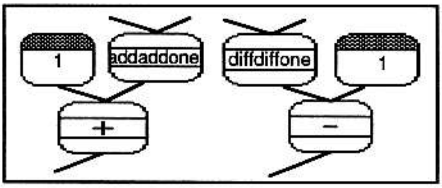


Figure 19: Two ABSYNT fragments

  b) The following two corresponding composites were plausible and thus moved into POSS:

C1: gmr (sum (const(C), addaddone (S1, S2)), add-pop (C-cl, Addaddone -hop (P1, P2))):-
is_const (C), gmr (S1, P1), gmr (S2, P2).

C2: gmr (diff (diffdiffone (S1, S2), const(C)), sub-pop (Diffdiffone-hop (P1, P2), C-cl)) :-
gmr (S1, P1), gmr (S2, P2), is_const (C).

("add-pop" is the primitive ABSYNT operator "+", "sub-pop" is the primitive ABSYNT operator "-",

"Addaddone-hop" and "Diffdiffone-hop" are self-defined "higher" ABSYNT operators with names given by the user.)

Furthermore, C1 was composed from the node rules:

O2: gmr(sum(S1, S2), add-pop(P1, P2)) :- gmr(S1, P1), gmr(S2, P2).

L2: gmr(const(C), C-cl) :- is_const(C).

O3: gmr(addaddone(S1, S2), Addaddone-hop(P1, P2)) :-
gmr(S1, P1), gmr(S2, P2).

The composite C1 can be described by the formula $(O2_1 \bullet L2)_1 \bullet O3$.

In order to obtain a generalization of these two composites, first the two solution fragments have to be syntactically aligned by goal elaboration rules. For example, by using the goal elaboration rule

E2: gmr (sum (S1, S2), P) :- gmr (sum (S2, S1), P).

expressing commutativity of addition, together with O2, L2, and O3 the program fragment on the left of Figure 20 can be generated. This syntactically aligned program fragment corresponds to the composite $C1_{ex}$ ("ex" for exchange):

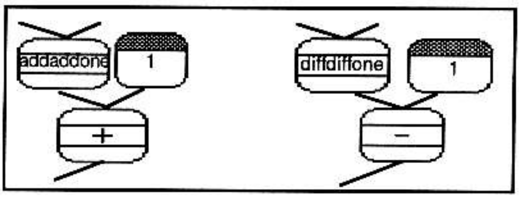


Figure 20: Syntactically aligned solution fragments of Figure 19

$C1_{ex}$: gmr (sum (addaddone (S1, S2), const(C)), add-pop(Addaddone-hop(P1, P2),C-cl)):-
gmr (S1, P1), gmr (S2, P2), is_const (C).

which is based on the same node rules as C1 and can be described by $(O2_1 \bullet O3)_1 \bullet L2$.

Now a new generalized rule G_{msg} can be created from C2 and $C1_{ex}$ by replacing the different goals and operators corresponding to the two program fragments (Figure 20) by variables. The possible values of the new variables are restricted by constraints. These constraints are built from the constants and their relations of the two original rules C2 and $C1_{ex}$.:

G_{msg}: gmr (Goal_1 (Goal_2 (S1, S2), const(C)),
Op_Name_1-pop (Op_Name_2-hop (P1, P2), C-cl)) :-
constraints([on(Goal_1, [sum, diff]),
on (Goal_2, [diffdiffone, addaddone]),
on(Op_Name_1, [add, sub]),

gmr(Goal_1(_,_), Op_Name_1(_,_)),
gmr(Goal_2(_,_), Op_Name_2(_,_))]),
gmr (S1, P1), gmr (S2, P2), is_const (C).

This is an example for a most specific generalization ("G_{msg}"). The rule G_{msg} is not able to parse or to generate similar problems. For example if the root goal is the goal to program a product the rule G_{msg} will fail, because the constraints are not satisfied. If the problem solver has no knowledge to program a product then there will be an impasse. One way to overcome this impasse would be to extend the constraints of the rule G_{msg} accordingly by inserting the "product" goal into the list [sum, diff] and the "mult" node into the list [add, sub].

It is also possible to generate another rule G_{mgg} from C2 and $C1_{ex}$. This most general generalization of the constraints differs from the example above by the missing variable restrictions:

G_{mgg}: gmr (Goal_1 (Goal_2 (S1, S2), const(C)),
Op_Name_1-pop (Op_Name_2-hop (P1, P2), C-cl)) :-
gmr (S1, P1), gmr (S2, P2), is_const (C).

This rule is an overgeneralization so it may produce errors. Remedial information (ie. error feedback to hypotheses) may lead to a stepwise restriction of the variables by constraints.

- As mentioned, introducing a *planning level* is another topic of our current research. Currently the learner's hypothetical solution plan is the parse tree of the solution. It is reconstructed retrospectively by the system after the solution is complete. We want the learner to be able to construct plans with an extension of the ABSYNT language by new goal nodes so that *mixed* ABSYNT programs containing operator nodes and goal nodes will be possible. The learner will be able to test hypotheses and to receive error and completion feedback at this *planning* level even if the learner has no idea yet about the implementation. Thus the learner may first *plan* a goal tree for the task at hand, test hypotheses about it, and debug it, if necessary. Afterwards the learner may *implement* the goals by replacing them with operator nodes or subtrees.

  For the user's point of view, the benefit of using goal nodes will be that hypotheses testing will be possible at the *planning stage*, not just at the implementation stage. From a psychological point of view, the benefit is that *objective* data about the planning process can be obtained in addition to the verbalizations. Finally, from a help system design point of view, the benefit is that in addition to hypotheses testing it will be possible to offer *planning rules* as help to the learner. The planning rules will be visual representations of GMR goal elaboration rules.

Computational feasibility and empirical validity. A current problem with the IM is that composites are first generated, based on the parse rules of a solution, and then tested for plausibility. Generation of composites can be time-consuming for very complicated ABSYNT program solutions. It is possible to change this situation by generating composites only for program fragments which were created by the student in temporal sequence. In this way many composites which would not pass the plausibility test would not be created in the first place.

Another problem is that the creation of the IM currently does not deal with program modifications performed by the student, like deleting and replacing nodes and links (although in the *evaluation* of the IM we were able to account for these data). Despite these shortcomings, we think that it is possible to extend the IM in appropriate ways. As we have also shown, it is possible to put the IM to empirical test and to draw conclusions for its improvement. For example, the study described above suggested changing simple parameter node rules. Some more testable hypotheses will be presented below. Thus advance towards an empirically validated knowledge diagnosis seems possible.

Adaptive help generation. The ultimate goal of the IM is to provide *adaptive* help or, more generally, to have an impact on the user-system-interaction in a way that takes account of the individual. In the ABSYNT Problem Solving Monitor, the need for the IM is very clear:

- There is a large solution space (the system is able to analyze and generate many solutions to given tasks) which is necessary because we want to be able to take care of novices' often unusual or unnecessarily complicated solutions (as illustrated in Figure 6).
- Because of the large solution space, there is usually a large amount of completion proposals that can be generated by the system. So the problem is which one to select. The task of the IM is to enable *user-centered* selection.

But as indicated, the role of the IM will not be restricted to the completion of ABSYNT nodes. Extending completion to the planning level and offering visual planning rules as help will impose additional demands on the IM. Additionally, the IM does more than just help selection. The information provided to the student may be varied in several ways, and this gives rise to empirical predictions which in turn might support or weaken the IM. Figure 21 illustrates how information intended as help can be varied, and what can be predicted. Basically, when the student is caught in an impasse and asks for a completion proposal, according to the IM there are two possible situations:

- The student has implementation knowledge but does not make use of it. Thus with respect to the interaction of planning and coding described earlier, there is a *planning* problem.
- The student lacks implementation knowledge, so there is a *coding* problem.

The latter, hypothetical situation is depicted in Figure 21: the student has just performed some programming actions, then gets stuck, and asks for completion proposals. According to the IM, there is a knowledge gap on the coding level, and after filling it the student would be able to proceed (shaded part of the horizontal arrow in the upper right of Figure 21). Now there are several possibilities to react to the gap: the information provided might vary in *grain size* and *amount* (on the left of Figure 21).

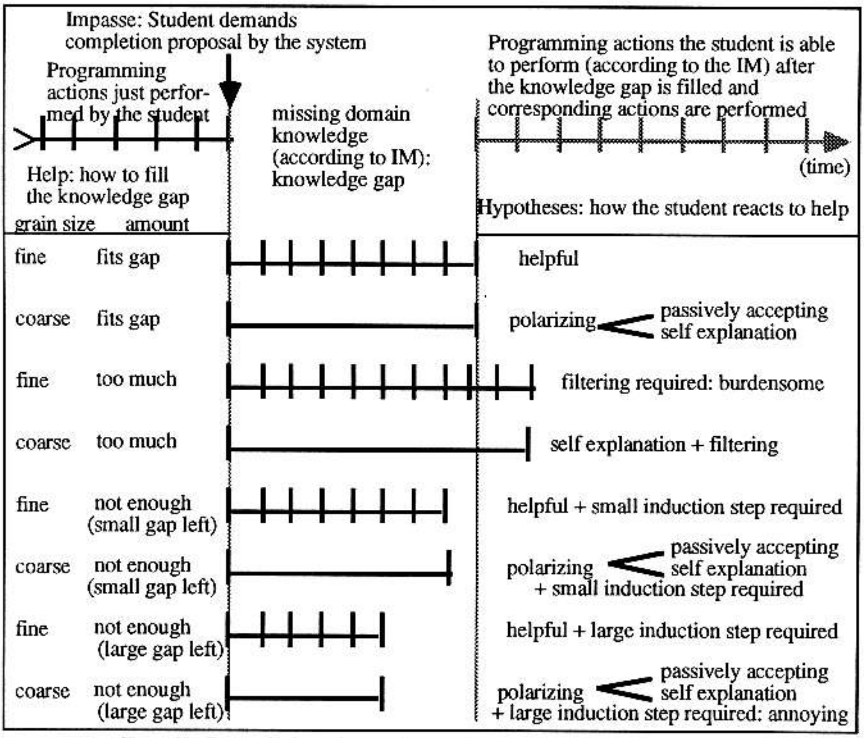


Figure 21: Types of information possibly provided in response to a knowledge gap diagnosed by the IM, and hypotheses concerning the student's reaction to this information

- *Grain size* concerns the rules underlying the completion proposal. If the grain size is *fine*, then the completion proposal may rest on a chain of simple rules which covers the gap. In this case the completion proposal may consist of an ABSYNT subtree with an explanation of each programming step needed to construct this subtree, where the explanation is based on the goal structure of the chain of simple rules. If the grain size is *coarse*, then the completion proposal may rest on a single composite (to take the other extreme). Thus the same subtree may be provided, but only with an explanation of the root goal.

- *Amount* concerns the relation between the completion and the gap. The completion proposal might *exactly fill* the gap, so subsequently the student can proceed by relying on her / his own knowledge. Alternatively, the completion proposal may contain *too much* information (more than necessary) or *not enough* information (the gap is not completely covered).

On the left and middle part of Figure 21, the different combinations of grain size and amount of information are shown. They lead to different hypotheses (on the right of Figure 21). We will describe some of them:

- If the information is fine-grained and exactly fills the gap (first row in Figure 21), then we would expect that the student considers this information as *helpful.*
- If the information is coarse-grained and exactly fills the gap (second row), then the student misses explanations. So s/he might either *passively accept* what is being offered, or engage in *self-explanation* [59].
- If the information is fine-grained but exceeds the knowledge gap (third row), then the student has to *"filter"* the content relevant to the current situation. This might be experienced as *burdensome.*
- If the information leaves a small knowledge gap (fifth and sixth row), then the student might try to induce one new simple rule and thereby cover the rest of the gap. (This situation seems similar to the induction of one subprocedure at a time by van Lehn's SIERRA program [55].)
- Finally, the last case to be considered here is that there is a large gap left, and the information offered is too coarse (last row). The student should experience such information as very inadequate to his current problem. Thus she or he should feel annoyed or even upset.

There remains much work, of course, to work out these hypotheses and put them to empirical test. But we think we have shown that the IM is an empirically fruitful approach to knowledge diagnosis and adaptive help generation which is testable and also touches upon further important research problems, like motivation and emotion.

Appendix A: GMR Rules of Figure 16 (PROLOG notation)

<u>IM rules</u>

Simple rules:

if-then-else node rule

```
gmr(branching(IF, THEN, ELSE), if-pop(P1, P2, P3)) :-
    gmr(IF, P1), gmr(THEN, P2), gmr(ELSE, P3).
```

less_than node rule

```
gmr(less_than(S1, S2), lt-pop(P1, P2)) :- gmr(S1, P1), gmr(S2, P2).
```

sum node rule

```
gmr(sum(S1, S2), add-pop(P1, P2)) :- gmr(S1, P1), gmr(S2, P2).
```

difference node rule

```
gmr(diff(S1, S2), sub-pop(P1, P2)) :- gmr(S1, P1), gmr(S2, P2).
```

parameter node rule

```
gmr(parm(P), P-pl) :- is_parm(P).
```

constant node rule

```
gmr(const(C), C-cl) :- is_const(C).
```

Schemata:

less_than & if-then-else composite

```
gmr(branching(less_than(S1,S2), S3, S4), if-pop(lt-pop(P1, P2), P3, P4)) :-
  gmr(S1, P1), gmr(S2, P2), gmr(S3, P3), gmr(S4, P4).
```

(less_than & if-then-else) & parameter & constant composite

```
gmr(branching(less_than(parm(Y), const(C)), parm(X), S),
        if-pop(lt-pop(Y-pl, C-cl), X-pl, P)) :-
    is_parm(Y), is_const(C), is_parm(X), gmr(S, P).
```

step_down_one & difference node-rule composite

```
gmr(step_down_one(S), sub-pop(P1,P2)) :- gmr(S,P1), gmr(const(1), P2).
```

Cases:

(less_than & if-then-else) & difference & change_sign plan-rules & product & difference node-rules & parameter & constant composite

```
gmr(branching(less_than(parm(A), parm(B)), diff(parm(C), parm(D)),
            diff(parm(E), parm(F))),

if-pop(lt-pop(A-pl, B-pl), mult-pop(sub-pop(D-pl, C-pl), -1-cl),
            sub-pop(E-pl, F-pl))) :-

    is_parm(A), is_parm(B), is_parm(D), is_parm(C),is_const(-1),
    is_parm(E), is_parm(F).
```

difference node-rule & parameter & constant composite

```
gmr(diff(parm(X), const(C)), sub-pop(X-pl, C-cl)) :- is_parm(X), is_const(C).
```

(step_down_one & difference) & parameter & constant composite

```
gmr(step_down_one(parm(X)), sub-pop(X-pl, 1-cl)) :- is_parm(X), is_const(1).
```

<u>EXPERT rules</u>

product node rule

gmr(product(S1, S2), mult-pop(P1, P2)) :- gmr(S1, P1), gmr(S2, P2).

difference plan rule

gmr(diff(S1, S2), ProgTree) :- gmr(ch_sign(diff(S2, S1))).

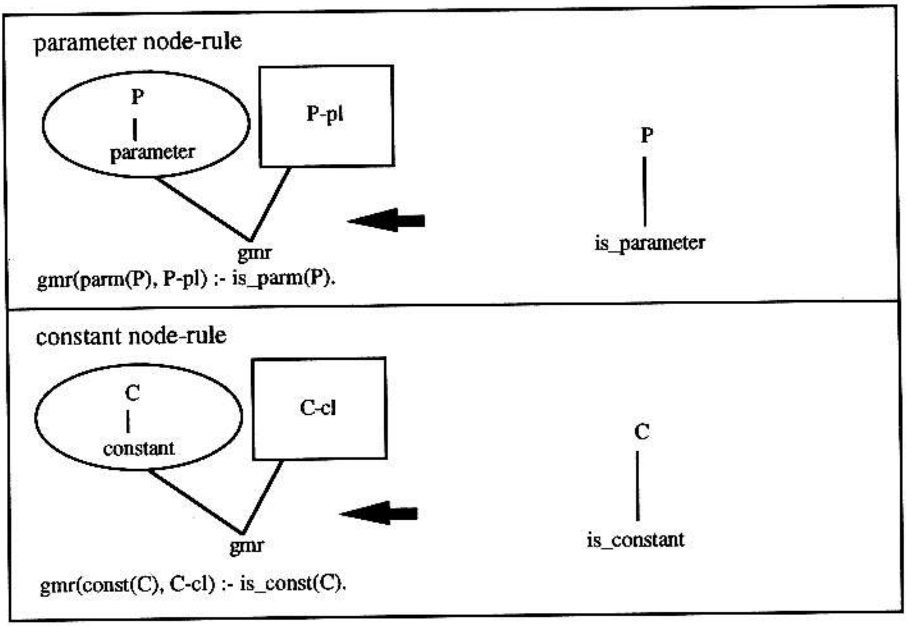
change_sign plan rule

gmr(ch_sign(S), ProgTree) :- gmr(product(S, const(-1))).

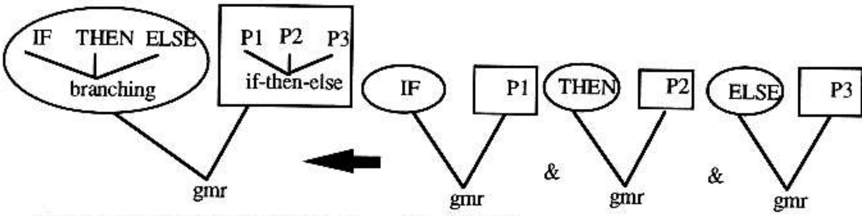
step_down_one plan rule

gmr(step_down_one(S), ProgTree) :- gmr(diff(S, const(-1))).

Appendix B: GMR Rules of Figures 17 and 18 (Visual and PROLOG notation)

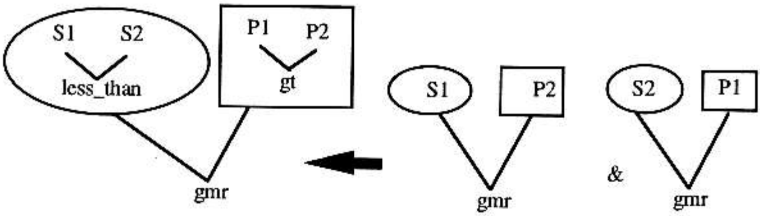


if-then-else node-rule



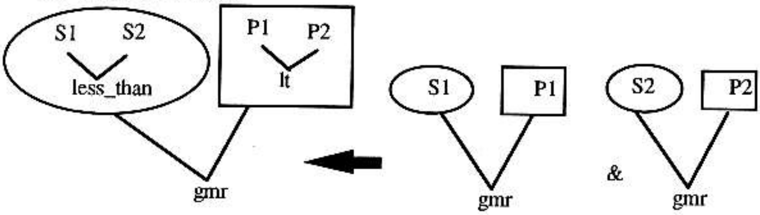
gmr(branching(IF, THEN, ELSE), ite-pop(P1, P2, P3)) :-
gmr(IF, P1), gmr(THEN, P2), gmr(ELSE, P3).

less_than node-rule(gt)



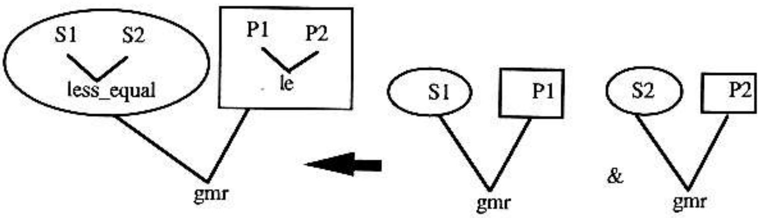
gmr(less_than(S1, S2), gt-pop(P1, P2)) :- gmr(S1, P2), gmr(S2, P1).

less_than node-rule(lt)



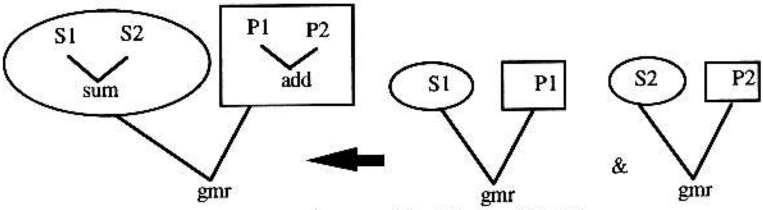
gmr(less_th(S1, S2), lt-pop(P1, P2)) :- gmr(S1, P1), gmr(S2, P2).

less_equal node-rule



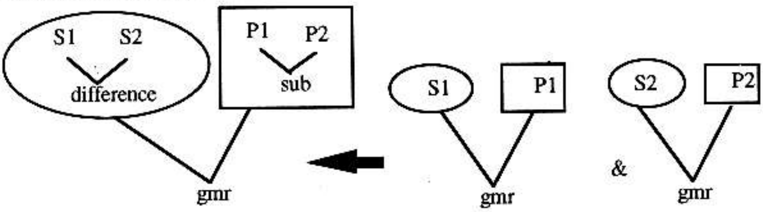
gmr(less_eq(S1, S2), le-pop(P1, P2)) :- gmr(S1, P1), gmr(S2, P2).

sum node-rule



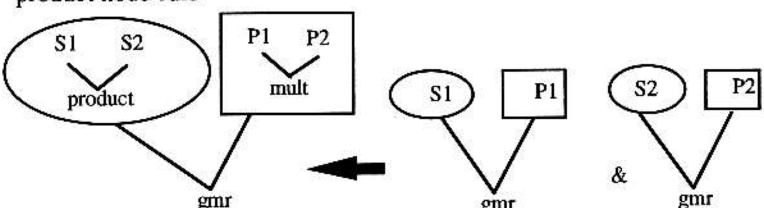
gmr(sum(S1, S2), add-pop(P1, P2)) :- gmr(S1, P1), gmr(S2, P2).

difference node-rule



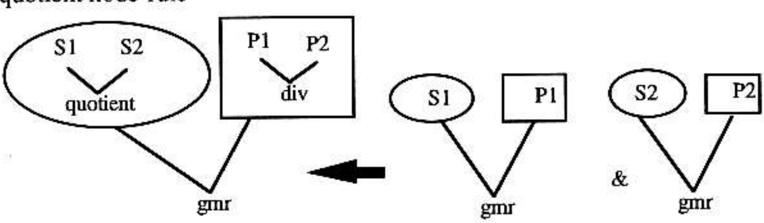
gmr(diff(S1, S2), sub-pop(P1, P2)) :- gmr(S1, P1), gmr(S2, P2).

product node-rule



gmr(product(S1, S2), mult-pop(P1, P2)) :- gmr(S1, P1), gmr(S2, P2).

quotient node-rule



gmr(quotient(S1, S2), div-pop(P1, P2)) :- gmr(S1, P1), gmr(S2, P2).

References

1. Anderson, J. R.: *The Architecture of Cognition.* Cambridge, MA: Harvard University Press 1983
2. Anderson, J. R.: Knowledge compilation: the general learning mechanism. In: *Machine Learning* (R. S. Michalski, J. G. Carbonell, & T. M. Mitchell, eds.). Vol. II, pp. 289-310. Los Altos, CA: Morgan Kaufmann 1986
3. Anderson, J. R.: Causal analysis and inductive learning. Proceedings of the 4th Workshop on Machine Learning, Irvine, CA, pp. 288-299 1987
4. Anderson, J. R.: A theory of the origins of human knowledge. *Artificial Intelligence*, 40, pp. 313-351 (1989)
5. Anderson, J. R., Boyle, C. F., Farrell, R., & Reiser, B. J.: Cognitive principles in the design of computer tutors. In: *Modelling Cognition* (P. Morris, ed.), pp. 93-133. New York, NY: Wiley 1987
6. Anderson, J. R., Conrad, F. G., & Corbett, A. T.: Skill acquisition and the LISP tutor. *Cognitive Science*, 13, pp. 467-505 (1989)
7. Anderson, J. R. & Swarecki, E.: The automated tutoring of introductory computer programming. *Communications of the ACM*, 29, pp. 842-849 (1986)
8. Bauer, F. L. & Goos, G.: *Informatik, Vol. 1.* 3rd ed. Berlin: Springer-Verlag 1982
9. Brown, J. S. & Burton, R. R.: Diagnosing bugs in a simple procedural skill. In: *Intelligent Tutoring Systems* (D. Sleeman & J. S. Brown, eds.). pp. 157-183. New York, NY: Academic Press 1982
10. Brown, J. S. & VanLehn, K.: Repair theory: a generative theory of bugs in procedural skills. *Cognitive Science*, 4, pp. 379-426 (1980)
11. Carbonell, J. G. & Gil, Y.: Learning by experimentation. Proceedings of the 4th Workshop on Machine Learning, Irvine, CA, pp. 256-266, 1987
12. Chang, S. K.(ed): *Principles of Visual Programming Systems.* Englewood Cliffs: Prentice Hall 1990
13. Chase, N. G. & Simon, H. A.: Perception in chess. *Cognitive Psychology*, 4, pp. 55-81 (1973)
14. Corbett, A. T. & Anderson, J. R.: Student modeling and mastery learning in a computer-based programming tutor. In: *Intelligent Tutoring Systems* (C. Frasson, G. Gauthier, & G. I. McCalla, eds.). Proceedings of the ITS 92, pp. 413-420, Lecture Notes in Computer Science, Vol. 608, Berlin: Springer-Verlag 1992
15. Egan, D. E. & Greeno, J. G.: Theory of rule induction: knowledge acquired in concept learning, serial pattern learning, and problem solving. In: *Knowledge and Cognition* (L. W. Gregg, ed.). pp. 43-103. Potomac: Lawrence Erlbaum Associates 1974
16. Elio, R. & Scharf, P. B.: Modeling novice-to-expert shifts in problem solving strategy and knowledge organization. *Cognitive Science*, 14, 579-639 (1990)
17. Ericsson, K. A. & Simon, H. A.: *Protocol Analysis.* Cambridge, MA: MIT Press 1984
18. Frasson, C. & Gauthier, G. (eds): *Intelligent Tutoring Systems.* Norwood, NJ: Ablex 1990
19. Gollwitzer, P. M.: Action phases and mind sets. In: *Handbook of Motivation and Cognition: Foundations of Social Behavior* (E. T. Higgins & R. M. Sorrentino, eds.). Vol. 2, pp. 53-92. 1990
20. Gollwitzer, P. M.: *Abwägen und planen.* Göttingen: Verlag für Psychologie 1991
21. Greer, J., McCalla, G. I., & Mark, M. A.: Incorporating granularity-based recognition into SCENT. In: *Proceedings of the 4th International Conference on Artificial Intelligence and Education*, Amsterdam (D. Bierman, J. Breuker & J. Sandberg, eds.), pp. 107-115, Amsterdam: IOS 1989
22. Gregory, S.: *Parallel logic Programming in PARLOG: The language and its Implementation.* Wokingham: Addison-Wesley 1987
23. Gugerty, L. & Olson, G. M.: Comprehension differences in debugging by skilled and novice programmers. In: *Empirical Studies of Programmers* (E. Soloway & S. Iyengar, eds.). pp. 13-27. Norwood, NJ: Ablex 1986
24. Hogger, Ch. J.: *Essentials of Logic Programming*. Oxford University Press 1990
25. Huber, P., Jensen, K., & Shapiro, R. M.: Hierarchies in colored petri nets. In: *Advances in Petri Nets 1990* (G. Rozenberg, ed.), Lecture Notes in Computer Science, Vol. 483, pp. 373-341, Berlin: Springer-Verlag 1991
26. Kearsley, G.: *Online Help Systems.* Norwood, NJ: Ablex 1988
27. Kowalski, R.: *Logic for Problem Solving*. Amsterdam: Elsevier 1979
28. Laird, J. E., Rosenbloom, P. S., & Newell, A.: *Universal Subgoaling and Chunking: The Automatic Generation and Learning of Goal Hierarchies.* Boston, MA: Kluwer 1986
29. Laird, J. E., Rosenbloom, P. S., & Newell, A.: SOAR: an architecture for general intelligence, *Artificial Intelligence*, 33, pp. 1-64 (1987)
30. Lanzetta, J. T. & Driscoll, J. M.: Effects of uncertainty and importance on information search in decision making. *Journal of Personality and Social Psychology*, 10, pp. 479-486 (1968)

31. Lehnert, W. G.: *The Process of Question Answering*. Hillsdale, NJ: Lawrence Erlbaum Associates 1978
32. Lewis, C.: Composition of productions. In: *Production System Models of Learning and Development* (D. Klahr, P. Langley, & R. Neches, eds.). pp. 329-358. Cambridge, MA: MIT Press 1987
33. McCalla, G. I., Greer, J., and the SCENT Research Team: Intelligent advising in problem solving domains: the SCENT-3 architecture. *Proceedings of the 2nd International Conference on Intelligent Tutoring Systems*, Montreal, Quebec (C. Frasson, G. Gauthier & G. McCalla, eds.), pp. 124-131, Lecture Notes in Computer Science, Vol. 608, Berlin: Springer-Verlag 1992
34. McCalla, G. I., Greer, J., & Coulman, R.: Enhancing the robustness of model-based recognition. *Proceedings of the 3rd International Workshop on User Modeling*, Wadern, Germany (E. Andre, R. Cohen, W. Graf, B. Kass, C. Paris, & W. Wahlster, eds.). pp. 240-248, Kaiserslautern: DFKI 1992
35. Minton, S. & Carbonell, J. G.: Strategies for learning search control rules: an explanation-based approach. Proceedings of the IJCAI 1987, pp. 228-235, Milan, Italy 1987
36. Möbus, C.: Toward the design of adaptive instructions and help for knowledge communication with the problem solving monitor ABSYNT. In: *Artificial Intelligence in Higher Education*, (V. Marik, O. Stepankova, & Z. Zdrahal, eds.), pp. 138-145, Lecture Notes in Computer Science, Vol. 451, Berlin: Springer-Verlag 1990
37. Möbus, C.: The relevance of computational models of knowledge acquisition for the design of helps in the problem solving monitor ABSYNT. Proceedings of the Conference on Advanced Research on Computers in Education (ARCE '90), pp. 137-144, (R. Lewis & O. Setsuko, eds.). Amsterdam: North-Holland 1991
38. Möbus, C. & Schröder, O.: Representing semantic knowledge with 2-dimensional rules in the domain of functional programming. In: *Visualization in Human-Computer Interaction, 7th Interdisciplinary Workshop in Informatics and Psychology*, Schärding. Austria, 1988 (P. Gorny & M. Tauber, eds.). pp. 47-81, Lecture Notes in Computer Science, Vol. 439, Berlin: Springer-Verlag 1990
39. Möbus, C., Schröder, O., & Thole, H. J.: Runtime modeling the novice-expert shift in programming skills on a rule-schema-case continuum. Proceedings of the IJCAI Workshop W.4 Agent Modelling for Intelligent Interaction, 12th International Joint Conference on Artificial Intelligence, (J. Kay & A. Quilici, eds.). pp. 137-143, Darling Harbour, Sydney, Australia, 1991
40. Möbus, C., Schröder, O., & Thole, H. J.: Online modelling the novice-expert shift in programming skills on a rule-schema-case partial order. In: *Working Notes of the AAAI Spring Symposium Series: Cognitive Aspects of Knowledge Acquisition*, pp. 155-161. Stanford University, 1992
41. Möbus, C. & Thole, H. J.: Tutors, instructions and helps. In: *Künstliche Intelligenz KIFS 1987* (Th. Christaller, ed.). pp. 336-385. Informatik-Fachberichte 202, Berlin: Springer-Verlag 1989
42. Möbus, C. & Thole, H. J.: Interactive support for planning visual programs in the problem solving monitor ABSYNT: giving feedback to user hypotheses on the language level. In: *Computer Assisted Learning*. Proceedings of the 3rd International Conference on Computer-Assisted Learning (ICCAL 90), Hagen, Germany, 1990. (D. H. Norrie & H. W. Six, eds.). pp. 36-49, Lecture Notes in Computer Science, Vol. 438, Berlin: Springer-Verlag 1990
43. Neves, D. M. & Anderson, J. R.: Knowledge compilation: mechanisms for the automatization of cognitive skills. In: *Cognitive Skills and their Acquisition* (J. R. Anderson, ed.). pp. 57-84. Hillsdale, NJ: Lawrence Erlbaum Associates 1981
44. Newell, A.: The knowledge level. *Artificial Intelligence*, 18, pp. 87-127 (1982)
45. 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*, 47, pp. 289-305 (1991)
46. Rosenbloom, P. S. & Newell, A.: Learning by chunking: a production system model of practice. In: *Production System Models of Learning and Development* (D. Klahr, P. Langley, & R. Neches, eds.). pp. 221-286. Cambridge, MA: MIT Press 1987
47. Schank, R. C.: *Dynamic Memory*. London: Cambridge University Press 1982
48. Schröder, O.: A model of the acquisition of rule knowledge with visual helps: the operational knowledge for a functional, visual programming language. In: *Computer Assisted Learning*, Proceedings of the 3rd International Conference on Computer-Assisted Learning (ICCAL 90), Hagen, Germany 1990 (D. H. Norrie & H. W. Six, eds.). pp. 142-157, Lecture Notes in Computer Science, Vol. 438, Berlin: Springer-Verlag 1990
49. Schröder, O.: *Erwerb von regelwissen mit visuellen hilfen: das semantikwissen für eine graphische funktionale programmiersprache*. Frankfurt: Lang 1992
50. Self, J. A.: Bypassing the intractable problem of student modeling. In: *Intelligent Tutoring Systems : at the Crossroads of Artificial Intelligence and Education* (C. Frasson & G. Gauthier, eds.). pp. 107-123. Norwood, NJ: Ablex 1990
51. Self, J. A.: Formal Approaches to Learner Modelling. Technical Report AI-59, Department of Computing, Lancaster University, Lancaster, UK 1991

52. Simon, H. A. & Simon, D. P.: Individual differences in solving physics problems. In: *Childrens' Thinking: What Develops?* (R. S. Siegler, ed.). pp. 325-348. Hillsdale, NJ: Lawrence Erlbaum Associates 1978
53. Sleeman, D.: An attempt to understand students' understanding of basic algebra. *Cognitive Science*, 8, pp. 387-412 (1984)
54. Sleeman, D. & Brown, J. S.: *Intelligent Tutoring Systems*. New York, NY: Academic Press 1982
55. VanLehn, K.: Learning one subprocedure per lesson. *Artificial Intelligence*, 31, pp. 1-40 (1987)
56. VanLehn, K.: Toward a theory of impasse-driven learning. In: *Learning Issues for Intelligent Tutoring Systems* (H. Mandl & A. Lesgold, eds.). pp. 19-41. New York, NY: Springer-Verlag 1988
57. VanLehn, K.: Learning events in the acquisition of three skills. Proceedings of the 11th Conference of the Cognitive Science Society, Ann Arbor, MI, pp. 434-441, 1989
58. VanLehn, K.: *Mind Bugs: The Origins of Procedural Misconceptions*. Cambridge,MA: MIT Press 1990
59. VanLehn, K.: Two pseudo-students: applications of machine learning to formative evaluation. In: *Advanced Research on Computers in Education*, (R. Lewis & S. Otsuki, eds.). pp 17-25. Amsterdam: Elsevier/IFIP 1991
60. VanLehn, K.: Rule acquisition events in the discovery of problem solving strategies. *Cognitive Science*, 15, pp. 1-47 (1991)
61. Vere, S. A.: Relational production systems. *Artificial Intelligence*, 8, pp. 47-68 (1977)
62. Weber, G.: Cognitive diagnosis and episodic modeling in an intelligent LISP tutor. *Proceedings of the 2nd International Conference on Intelligent Tutoring Systems*, Montreal, Quebec (C. Frasson, G. Gauthier & G. McCalla, eds.), pp. 207-214, Lecture Notes in Computer Science, Vol. 608, Berlin: Springer-Verlag 1992
63. Weber, G.: Explanation-based retrieval in a case-based learning model. Proceedings of the 13th Annual Meeting of the Cognitive Science Society, Chicago, IL, pp. 522-527, Hillsdale, NJ: Lawrence Erlbaum Associates 1991
64. Wenger, E.: *Artificial Intelligence and Tutoring Systems*. Los Altos, CA 1987
65. Wolff, J. G.: Cognitive development as optimization. In: *Computational Models of Learning* (L. Bolc, ed.). pp. 161-205. Berlin: Springer-Verlag 1987

Student Modelling: The Key to Individualized Knowledge-Based Instruction

Edited by

Jim E. Greer

Gordon I. McCalla

Department of Computational Science
1C101 Engineering Building
University of Saskatchewan
Saskatoon, Saskatchewan
Canada S7N 0W0

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 "Student Modelling: The Key to Individualized Knowledge-Based Instruction", held in Ste. Adele, Quebec, Canada, May 4–8, 1991

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

ISBN 3-540-57510-3 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-57510-3 Springer-Verlag New York Berlin Heidelberg

CIP data applied for.

Printed in Germany

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

Student Modelling: The Key to Individualized Knowledge-Based Instruction

Edited by

Jim E. Greer   Gordon I. McCalla

NATO ASI Series

Series F: Computer and Systems Sciences, Vol. 125

NATO ASI Series

Springer-Verlag
Berlin Heidelberg New York London Paris Tokyo
Hong Kong Barcelona Budapest

ISBN 3-540-57510-3 · ISBN 0-387-57510-3