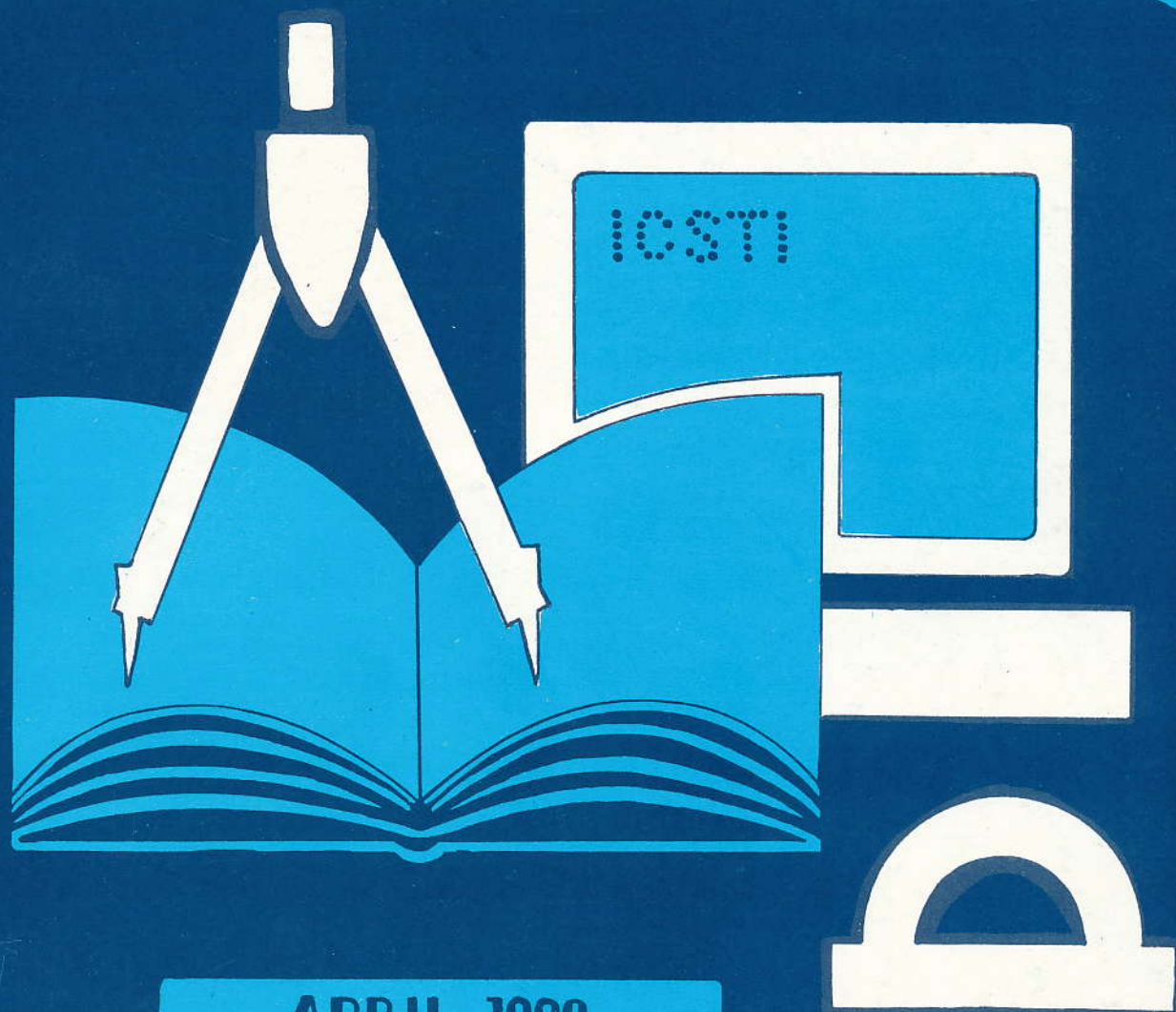




КОНФЕРЕНЦИЯ ВОСТОК-ЗАПАД
ПО НОВЫМ ИНФОРМАЦИОННЫМ ТЕХНОЛОГИЯМ
В ОБРАЗОВАНИИ

EAST-WEST CONFERENCE
ON EMERGING COMPUTER TECHNOLOGIES
IN EDUCATION

ДОКЛАДЫ PROCEEDINGS



APRIL 1992
• MOSCOW •

ON-LINE MODELLING THE NOVICE-EXPERT SHIFT IN PROGRAMMING SKILLS ON A RULE-SCHEMA-CASE PARTIAL ORDER FOR INTERNAL STUDENT MODELS

Claus Moebus, Olaf Schroeder, Heinz-Juergen Thole

University of Oldenburg,
D-2900 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. This is necessary because especially novices often produce "unusual" solutions. On the other hand, it requires a model of the students' actual state of domain knowledge in order to provide user-centered help. The model distinguishes between knowledge acquisition and knowledge improvement. *Knowledge acquisition* is represented by augmenting the model with expert planning knowledge represented as rules. The acquisition of malrules is possible, too. *Knowledge improvement* is represented by rule composition. In this way, the knowledge contained in the model can be located on a partial order from general rules to more specific schemas for solution fragments to specific cases (= example solutions).

Introduction

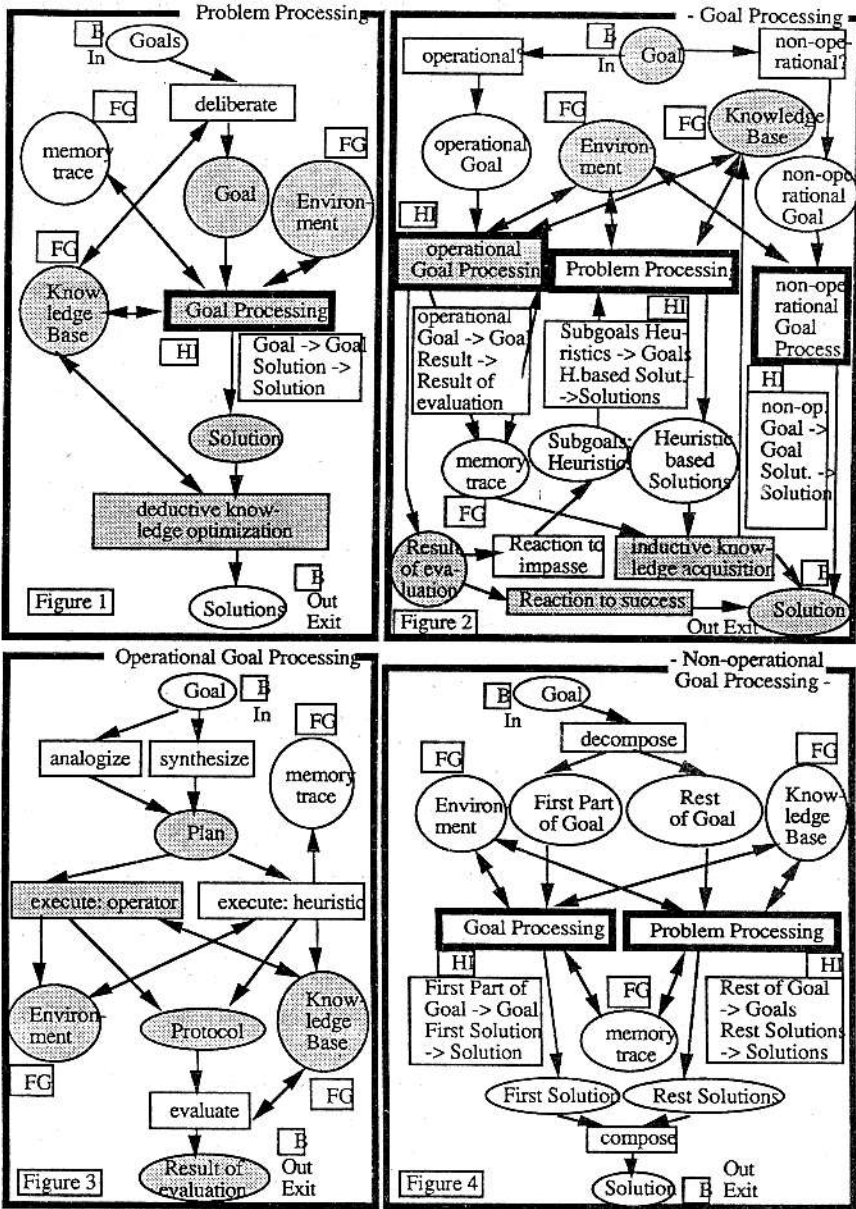
Modelling knowledge acquisition processes has been recognized as an important research topic (Anderson, 1983, 1986, 1989; Brown & Burton, 1982; Brown & VanLehn, 1980; Rosenbloom & Newell, 1986, 1987; Rosenbloom et al., 1991; Sleeman, 1984) to answer questions like: Which order is the best for a set of tasks to be worked on? Why is information useless to one person and helpful to another? How is help material to be designed? Answering these questions requires hypotheses about the learner's knowledge states and knowledge acquisition processes. This is especially true within help and tutoring systems (Frasson & Gauthier, 1990; Kearsley, 1988; Sleeman & Brown, 1982; Wenger, 1987), where online diagnosis of the learner's knowledge is necessary in order to react in an adequate way. This diagnosis has to be both efficient and valid. But to achieve both goals is a difficult problem (Self, 1990, 1991) because there is only a limited source of information, the learner's stream of actions.

We model the change of knowledge with two models: the *internal* model (IM) and the *external* model (EM). The IM (described in this paper) is an integrated part of the help system. Its purpose is to provide user-centered feedback to the stream of problem solving actions. The EM is not a part of the help system ("external" to it) but is designed to simulate the knowledge acquisition process of learners on a level of detail, including protocol analyses of verbal data, not available to the IM. The IM *describes* the hypothetical knowledge growth of the learner. The EM contains additional hypothetical control knowledge and thus provides hypothetical *reasons* for the knowledge changes described in the IM. One of its purposes is to support the development of the IM.

For modelling knowledge acquisition processes, a theoretical position concerning *problem solving and learning* is necessary which is able to describe the shift of a learner from novice to expert. We have integrated several approaches into a theory we call ISP-DL Theory (Impasse Success Problem solving Driven Learning Theory) to be described now.

The ISP-DL Knowledge Acquisition Theory

For the informal description of our ISP-DL theory we use *hierarchical higher Petri-nets* (Huber et al., 1990). The process is divided into 4 recursive subprocesses: "Problem Processing", "Goal Processing", "Nonoperational Goal processing" and "Operational Goal Processing" (Figures 1-4). Places represent states or data memories, transitions represent events or process steps. Places can 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 (B for border place, FG for global fusion set). A FG tagged place is common to several nets (eg. the Knowledge Base). Transitions can be tagged with HI (HI for hierarchical invocation transition).



Figures 1 - 4: THE ISP-DL Theory of problem solving and learning

This means that the process is continued in a freshly created instance of the subnet. Within the dotted boxes it is shown which places are corresponding in the calling net and the called net. Shaded transitions and places are of concern for the IM.

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 (Newell, 1982). More precisely, 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: goal is achieved if 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?".

In the latter case, the process is continued in the page "non- operational goal processing" (Figure 3). The problem can be decomposed and subsolutions are composed to a final solution.

In the former case processing is done according to the page "operational goal processing" (Figure 4). A plan is synthesized by applying problem solving operators, or it is created by analogical reasoning. The plan may be a partially ordered sequence or hierarchy of problem solving operators or heuristics. The PS is in favour of applying problem solving operators. If he uses heuristics a memory trace is kept. In either case, a problem solving protocol is generated, 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".

The reaction of the PS to success is: leave "Goal Processing" with a solution. On the other hand the reaction to an impasse is the creation of subgoals to use weak heuristics for problem solving. The corresponding problem solving process creates heuristic-based solutions and memory traces during the application of the heuristics. After that the PS will generate inductively a new operator on the basis of the memory trace and the knowledge about a success.

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 is the theoretically and empirically validated statement: *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.*

The ABSYNT Problem Solving Monitor

The ABSYNT problem solving monitor provides an iconic programming environment (Figure 5) and is aimed at supporting novices' acquisition of functional programming concepts up to recursive systems (Moebus, 1990; Moebus & Thole, 1990). The main components of ABSYNT are a visual editor, trace, and a help component: a hypotheses testing environment. In this environment, the learner may state hypotheses (bold parts of the program in the upper worksheet in Figure 5) about the correctness of total programs or parts thereof. The hypothesis is: "Is it possible to embed the boldly marked fragment of the program in a correct solution?". 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. The student's proposal (Figure 5, upper half) for the "even" problem does not terminate for odd arguments. Despite of that his/her hypothesis (upper half, bold) is embeddable in a correct solution. So the hypothesis is returned as feedback to the student (thin part of lower half of Figure 5). The student then may ask for a completion proposal generated by the system (bold part of lower half of Figure 5).

One reason for the hypotheses testing approach is that in functional programming a bug usually cannot be absolutely localized, and there is a variety of ways to debug a wrong solution. Hypotheses testing leaves this decision to the PS and thereby provides a rich data source about the learner's knowledge state.

The answers to the learner's hypotheses are generated by rules defining a goals-means-relation (GMR). These rules may be viewed as "pure" expert knowledge not influenced by learning. Thus we will call this set of rules EXPERT in the remainder of the paper. Currently, EXPERT contains 622 rules and analyzes and synthesizes several millions of solutions for 40 tasks (Moebus, 1990; 1991; Moebus & Thole, 1990).

For adaptive help generation, the EXPERT rules have to be augmented by an internal student model (IM). The function of the IM is to select the completion which is maximally consistent with the learner's current knowledge state. This should reduce the learner's surprise to a minimum.

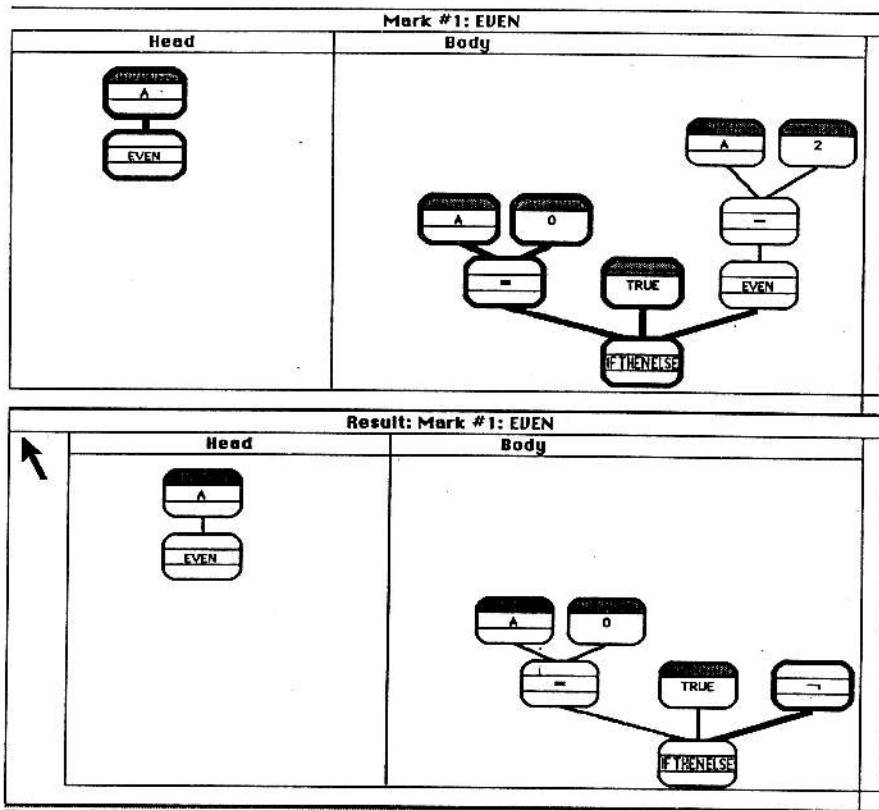


Figure 5: Snapshot of the ABSYNT hypotheses testing environment

GMR Rules

This section describes the GMR-rules which can be partitioned in two ways: rule type (simple, composed) and data base of the rules (EXPERT, POSS, IM). There are three kinds of simple rules: goal elaboration rules, rules implementing one ABSYNT node, and rules implementing ABSYNT program heads. 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 containing at least one variable which can be bound to a subtree is called a schema. If all variables in a composite can only be bound to node names or values, then the composite is called a case.

The other way to split the set GMR is the data base of the rules. EXPERT contains the expert domain knowledge. The sets IM and POSS are created at runtime and will be described below.

Figure 6 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 goal -implementation - pair where the goal is contained in the ellipse and the implementation is contained in the rectangle. The rule body contains one goal-implementation-pair or a conjunction of several pairs, or a primitive predicate.

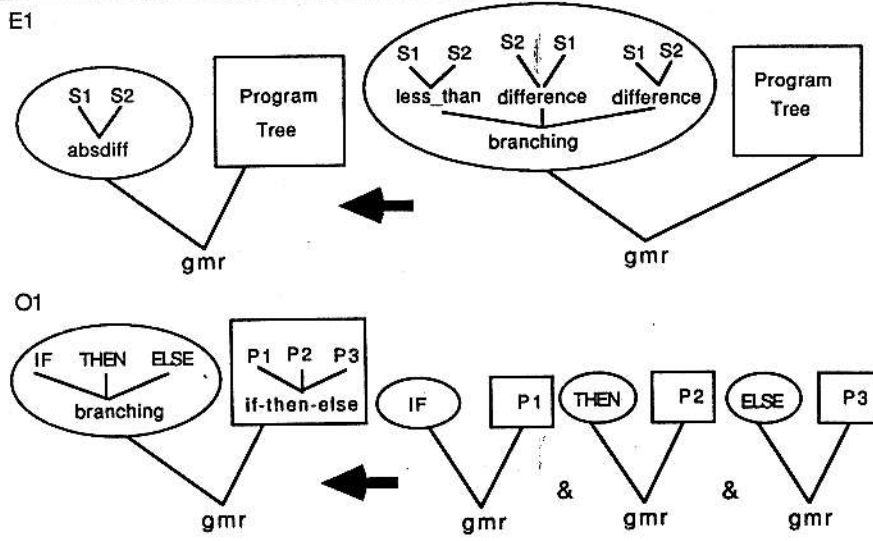


Figure 6: A goal elaboration rule (E1) and a rule implementing one ABSYNT node (O1)

The first rule of Figure 6, 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 6 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 with three connections leaving from this node, and leave space above these connections 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.

Composition of Rules

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

If we view the rules as Horn clauses (Kowalski, 1979), then the composite RIJ of two rules RI and RJ can be described by the inference rule:

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

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. s is the most general unifier of P and P'.

For example we can compose the schema C7 (Figure 7) out of the set of simple rules {O1, O5, L1, L2}, where:

- O1: $\text{gmr}(\text{branching}(\text{If}, \text{Then}, \text{Else}), \text{ite-pop}(\text{P1}, \text{P2}, \text{P3})) \text{- gmr}(\text{If}, \text{P1}), \text{gmr}(\text{Then}, \text{P2}), \text{gmr}(\text{Else}, \text{P3}).$
- O5: $\text{gmr}(\text{equal}(\text{S1}, \text{S2}), \text{eq-pop}(\text{P1}, \text{P2})) \text{- gmr}(\text{S1}, \text{P1}), \text{gmr}(\text{S2}, \text{P2}).$
- L1: $\text{gmr}(\text{parm}(\text{P}), \text{P-pl}) \text{- is_parm}(\text{P}).$
- L2: $\text{gmr}(\text{const}(\text{C}), \text{C-cl}) \text{- is_const}(\text{C}).$
- C7: $\text{gmr}(\text{branching}(\text{equal}(\text{parm}(\text{Y}), \text{const}(\text{C})), \text{parm}(\text{X}), \text{Else}), \text{ite-pop}(\text{eq-pop}(\text{Y-pl}, \text{C-cl}), \text{X-pl}, \text{P})) \text{- is_parm}(\text{Y}), \text{is_const}(\text{C}), \text{is_parm}(\text{X}), \text{gmr}(\text{Else}, \text{P}).$

- ite-pop = primitive ABSYNT operator "if-then-else"
- eq-pop = primitive ABSYNT operator "="
- P-pl = unnamed ABSYNT parameter node
- C-cl = empty ABSYNT constant node

C7 : Composite of the rules O1, O5, L1, and L2

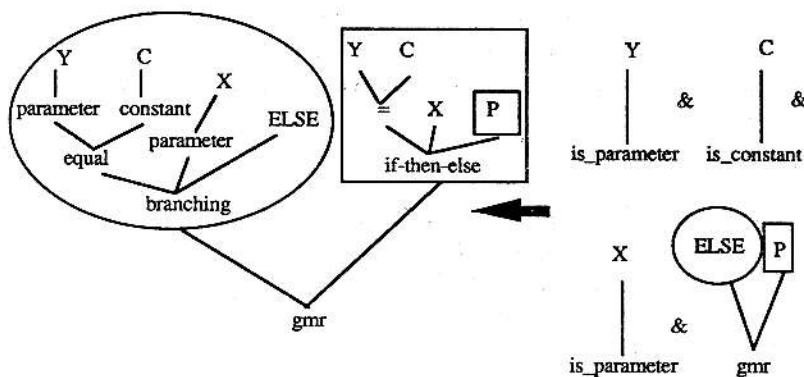


Figure 7: The composite C7

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

$$RIJ = RI_k \cdot RJ$$

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 (e.g.: O13 = Else). The semantics of "e" can be described in three steps. First, the variable in place k in the goal term 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 k-th variable and which unifies with the head of RJ is replaced by the body of RJ. Third the unifier s is applied to this term resulting in the composed rule RIJ.

For example O12 e L1 = $\text{gmr}(\text{branching}(\text{If}, \text{parm}(\text{P}), \text{Else}), \text{ite-pop}(\text{P1}, \text{P}, \text{P3})) \text{- gmr}(\text{If}, \text{P1}), \text{is_parm}(\text{P}), \text{gmr}(\text{Else}, \text{P3}).$ C7 can be composed out of the ruleset {O1, O5, L1, L2} in 12 different ways. Two possibilities are:

$$C7 = (O12 \cdot L1)_1 \cdot ((O52 \cdot L2)_1 \cdot L1)$$

or

$$C7 = (((O11 \cdot O5)_3 \cdot L1)_2 \cdot L2)_1 \cdot L1$$

Empirical Constraints of Simple Rules, Chains, Schemata and Cases

Rules, rule chains and schemata give rise to different empirical predictions. Novices work sequentially, set more subgoals, and need more control decisions, while experts work in parallel, set less subgoals, and need less control decisions. This difference is reflected in the partial order from simple rules to schemata to specific cases or solution examples.

We pose the following hypotheses:

- If the problem solver applies a rule which contains a goal tree and a program fragment in its head, then these goals may be verbalized and this fragment is implemented in a continuous uninterrupted action sequence. Verbalizations and actions are intermixed.

- If the problem solver applies a rule which contains subgoals in its body, then these subgoals may be verbalized.

Comparing the application of a composite to the application of the corresponding chain of simple rules, this leads to the following empirical consequences:

- For the composite, the order in which the parts of the program fragments in the rule head are implemented is indeterminate and not predictable. The same is true for the verbalized goals in the goal tree.
- For the rule chain, not only the set but also the order of programming actions is predictable.
- The composite is accompanied by verbalizations to a less degree. Cases should not be accompanied by verbalizations at all. For the rule chain, the content and the order of the verbalizations is predictable.
- Compared to the corresponding chain of simple rules, the program fragment in the head of the composite is programmed faster, because of the simpler goal structure of the composite and the smaller number of control decisions.

These relationships are illustrated in Figure 8 (suppressing the location information for compositions). The rule sets are organized in a partial order which reflects the degree of verbalization, performance time, and degree of predictability of the order of programming actions.

The application of chains of rules, which can be built from the rule sets containing simple rules and composites, and the schema C7 all lead to the same solution: the not yet finished ABSYNT program depicted in the head of C7. But we would expect differences in verbalizations and performance time. For example, the rule chain built out of the elements of the set {O1, O5, L1, L2} should be accompanied by more verbalizations and longer performance time than the other rule chains and C7 (in Figure 8).

For example the rule chain (O1, L1, O5, L2, L1) which when composed generates C7 according to (O1 · L1)1 · ((O52 · L2)1 · L1) leads to the prediction of the stream of events: events(O1) < events(L1) < events(O5) < events(L2) < events(L1), where:

- events(O1) = {verb(branching(;;)), act(if-then-else), act(link(if-then-else1,P1)), act(link(if-then-else2,P2)), act(link(if-then-else3,P3)), space(P1),space(P2), space(P3)},
- events(L1) = {verb(parameter(.)), act(parameter(X))},
- events(O5) = {verb(equal(.;)), act(=), act(link(=1, P1)), act(link(=2, P2)), space(P1), space(P2)}
- events(L2) = {verb(constant(.)), act(constant(C))}
- events(L1) = {verb(parameter(.)), act(parameter(X))}.

A < B means that the events of set A are followed by the events in set B.

The empirical meaning of the terms is:

- verb(Functor(., ..., .)): the value of Functor and the instantiated arguments of Functor are possibly verbalized
- act(Functor): the Functor will necessarily be implemented by an ABSYNT node
- act(Functor(.)): the Functor will be necessarily be implemented by an ABSYNT node which content is filled by the argument value
- act(link(Node1_i, Node2)): necessarily an ABSYNT link will be implemented between the i-th input of ABSYNT-Node1 and ABSYNT- Node2.
- space(.): necessarily a space in the programming environment will be reserved for the program fragment which is denoted by the argument of space.

The empirical predictions of the schema C7 are less constrained. The prediction is not a sequence of event sets but only one set of events:

- events(C7) = { verb(branching(;;)), verb(equal(.;)), verb(parameter(.)), verb(constant(.)), act(if-then-else), act(=), act(parameter(Y)), act(constant(C)), act(parameter(X)), act(link(if-then-else1,=)), act(link(if-then-else2,X)), act(link(if-then-else3,P)), act(link(=1,Y)),act(link(=2,C)), space(P)}.

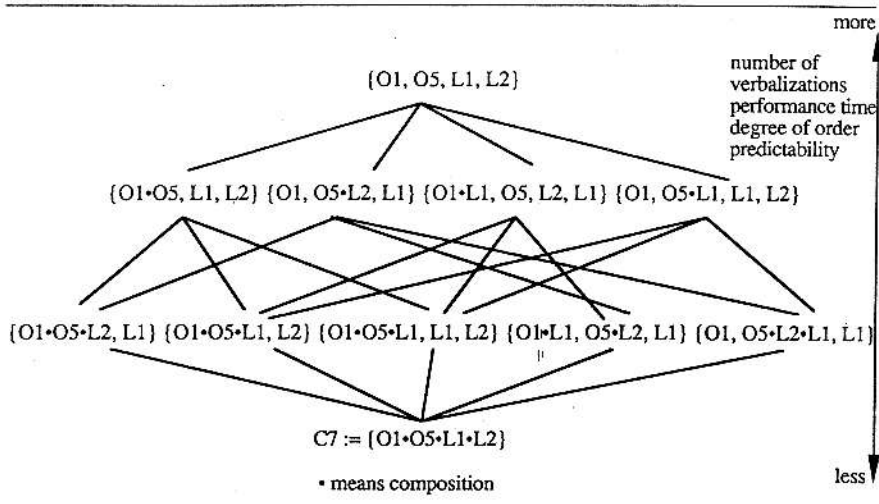


Figure 8: Rule sets partially ordered according to expected number of verbalizations, performance time, and degree of order predictability

Evolution of the IM during Problem Solving

The IM has the following general features:

- In accordance with ISP-DL-Theory, the IM contains simple rules representing acquired but not yet improved knowledge, and composites representing various degrees of expertise.
- Since knowledge improvement should result in sped-up performance, a composite becomes part of the 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 or the corresponding chain of simple rules.
- The IM contains only simple rules and composites which proved to be plausible with respect to an action sequence at least once. By this we mean the following:

Except for "goal elaboration rules", the simple rules and composites contain a program fragment in their rule head (Figures 6 and 7). Thus if the PS applies a certain rule from his domain knowledge, then we expect that he implements the program fragment in the head of the rule in an uninterrupted temporal sequence. The order of action steps within this sequence is indeterminate.

With respect to some sequence of actions, simple rules and composites form four subsets:

1. Rules not containing any program fragments ("goal elaboration rules") are nondecisive with respect to the action sequence. (But fragments of verbalizations can be related to the goal elaboration rules; Moebus & Thole, 1990).
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. 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 noncontinuous 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 number of action steps explained by this rule in the problem solving process of the PS. Thus the credit is determined by the length of the action sequence explained by the rule and the number of its successful applications.

- According to ISP-DL-Theory, a simple rule acquired by impasse-driven learning can only be improved after its successful application (success-driven learning). This implies for the IM that it cannot at the same time be augmented by a new simple rule and by composites built from this simple rule. Rather, the possible composites have to wait for incorporation in the IM. 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 in a generate-and-test-cycle and kept in POSS as candidates. Those surviving a test phase are then moved into the IM. So the IM contains only simple rules and composites for which we

hypothesize that the learner used them already, whereas POSS contains only composites which the PS might have created as a result of success-driven learning, but did not use them yet.

Figure 9 shows the development of the IM during the knowledge acquisition process.

Start (Top of Figure 9): Before performing the first task, both sets IM and POSS are empty.

$i := 1$: Now the learner solves the first task.

First Test: IM and POSS are empty, so nothing happens.

First Parse: The learner's solution to the first task is parsed with the EXPERT rules.

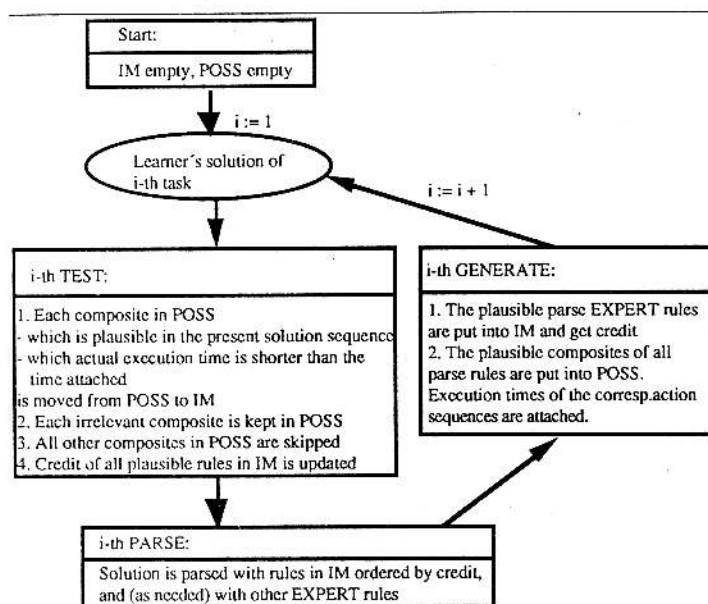


Figure 9: Development of the IM during the knowledge acquisition process

First Generate: The EXPERT rules just used for parsing are compared to the action sequence which produced the learner's solution, and which is saved in a log file. The plausible parse EXPERT rules are put into the IM and get credit. These rules are hypothesized as newly acquired by PS 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. They are candidates of improved knowledge useful for future tasks. For each plausible composite, the time needed by the PS to perform the corresponding action sequence is attached.

$i := i + 1$: Now the learner solves the second task.

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.

The composites meeting these requirements are put into the IM. Composites in POSS which are 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). Finally, the credits of all rules in the IM which are plausible with respect to the present action sequence are updated.

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.

An Empirical Example illustrating Plausibility

To illustrate, Figure 10 shows a continuous fragment of the action sequence of a PS, Subject 2 (S2), on a programming task. We will restrict our attention to the rules O1, O5, L1, L2, and C7 (see Figures 6 and 7). When S2 performs the sequence of Figure 10, 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 9) 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 10 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 10). 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 9). 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 9) starts. O5 is an EXPERT rule used for parsing. But O5 is implausible, since its corresponding actions were performed at 11:15:22, 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 7) is a composite formed from O1, O5, L1 and L2. This composite is plausible because it describes the uninterrupted sequence of programming actions from 11:15:08 to 11:16:55 (see Figure 10) - 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 now 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 10). This credit will be incremented by 13 each time the composite is plausible again.

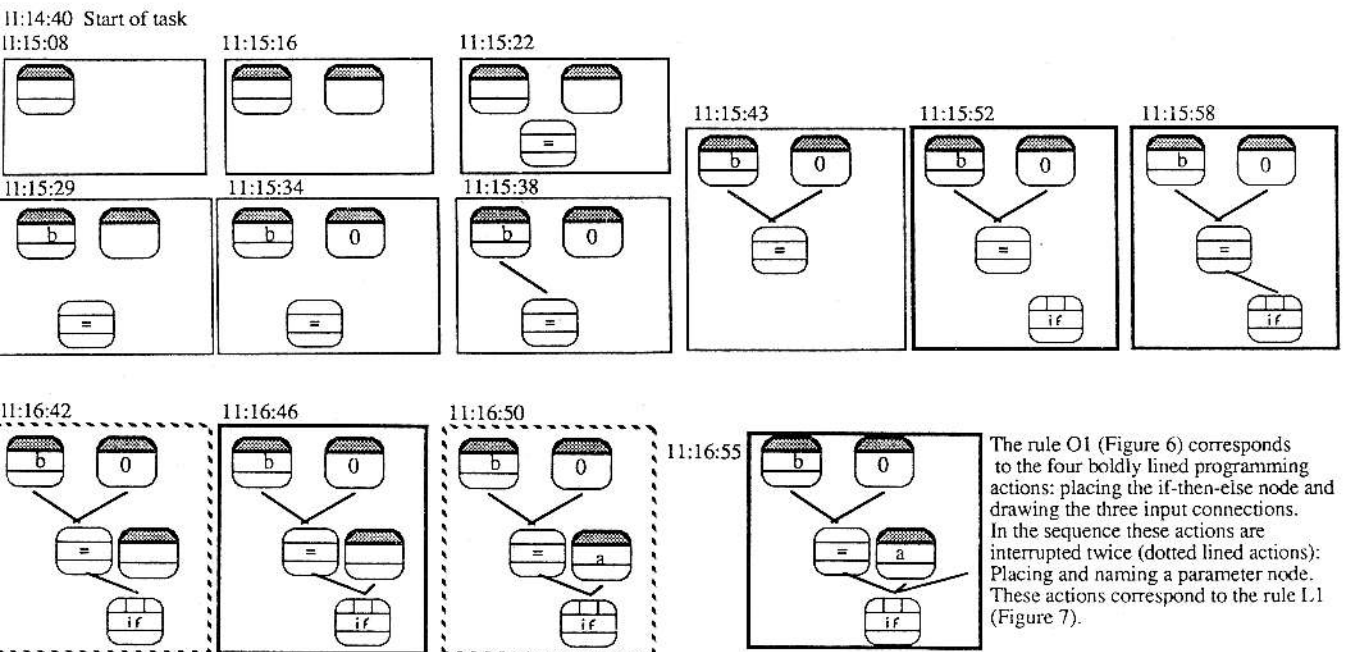


Figure 10: A continuous fragment of a sequence of programming actions of Subject S2

Outlook

Further empirical analyses (in contrast to e.g. Elio & Scharf, 1990) of solutions and continuous action streams are in progress. The same is true for the schema-based help generation. When the PS is caught in an impasse for a task *j* it is possible to use our hypotheses testing approach. We generate a solution proposal to task *j* based on schemata taken from the IM. These schemata represent the content of a hypothesis now generated by the IM and not by the user as before. Thus it is possible to offer help which uses episodic information and which was used by the PS successfully before.

References

- Anderson, J.R., *The Architecture of Cognition*. Harvard University Press, 1983
- Anderson, J.R., *Knowledge Compilation: The General Learning Mechanism*. In: Michalski, R.S.; Carbonell, J.G.; Mitchell, T.M., *Machine Learning*, Vol. II. Los Altos Kaufman, 1986, 289-310
- Anderson, J.R., *A Theory of the Origins of Human Knowledge*, *Artificial Intelligence*, 1989, 40, 313-351
- Brown, J.S., Burton, R.R., *Diagnosing Bugs in a Simple Procedural Skill*. In: Sleeman, D., Brown, J.S., *Intelligent Tutoring Systems*, New York: Academic Press, 1982, 157-183
- Brown, J.S., van Lehn, K., *Repair Theory: A Generative Theory of Bugs in Procedural Skills*. *Cognitive Science*, 1980, 4, 379-426 Elio, R., Scharf, P.B., *Modeling Novice-to-Expert Shifts in Problem Solving Strategy and Knowledge Organization*. *Cognitive Science*, 1990, 14, 579-639
- Frasson, C., Gauthier, G. (eds), *Intelligent Tutoring Systems*, Norwood, N.J.: Ablex, 1990
- Huber, P., Jensen, K. & Shapiro, R.M., *Hierarchies in Coloured Petri Nets*, in: G. Rozenberg (ed.): *Advances in Petri Nets 1990*, Lecture Notes of Computer Science, Heidelberg: Springer
- Kearsley, G., *Online Help Systems*. Norwood: Ablex, 1988 Kowalski, R., *Logic for Problem Solving*, Amsterdam: Elsevier Science Publ., 1979
- Moebus, C., *Toward the Design of Adaptive Instructions and Help for Knowledge Communication with the Problem Solving Monitor ABSYNT*. in: Marik, V.; Stepankova, O.; Zdrahal, Z. (eds): *Artificial Intelligence in Higher Education*, Proceedings of the CEPES UNESCO International Symposium Prague, CSFR, October 23 - 25, 1989, Berlin - Heidelberg - New York: Springer, Lecture Notes in Computer Science, No.451 (subseries LNAI), 1990, 138 - 145
- Moebus, C., *The Relevance of Computational Models of Knowledge Acquisition for the Design of Helps in the Problem Solving Monitor ABSYNT*, in: Lewis, R., Setsuko, O. (eds), *Advanced Research on Computers in Education (ARCE '90)*, Proceedings, Amsterdam: North- Holland, 1991, 137-144
- Moebus, 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: Norrie, D.H., Six, H.W. (ed), *Computer Assisted Learning*. Proceedings of the 3rd International Conference on Computer- Assisted Learning ICCAL 90, Hagen, Germany, Lecture Notes in Computer Science, Vol. 438, Heidelberg: Springer, 1990, 36-49
- Newell, A., *The Knowledge Level*. *Artificial Intelligence*, 1982, 18, 87-127
- Rosenbloom, P.S., Laird, J.E., Newell, A., McCarl, R., *A Preliminary Analysis of the SOAR Architecture as a Basis for General Intelligence*, *Artificial Intelligence*, 1991, 47, 289-305 Rosenbloom, P.; Newell, A., *The Chunking of Goal Hierarchies: A Generalized Model of Practice*. In: Michalski, R.S.; Carbonell, J.G.; Mitchell, T.M., *Machine Learning*, Vol. II. Los Altos: Kaufman, 1986, 247-288

Osin

- Rosenbloom, P.; Newell, A., Learning by Chunking: A Production System Model of Practice. In: Klahr, D.; Langley, P.; Neches, R. (eds), Production System Models of Learning and Development. Cambridge: MIT Press, 1987, 221-286
- Self, J.A., Bypassing the Intractable Problem of Student Modeling. In, Frasson, C., Gauthier, G. (eds), Intelligent Tutoring Systems, Norwood, N.J.: Ablex, 1990, 107-123
- Self, J.A., Formal Approaches to Learner Modelling. Technical Report AI-59, Dept. of Computing, Lancaster University, Lancaster, England, 1991
- Sleeman, D., An Attempt to Understand Students Understanding of Basic Algebra. Cognitive Science, 1984, 8, 387-412
- Sleeman, D., Brown, J.S., Intelligent Tutoring Systems, New York: Academic Press, 1982
- Wenger, E., Artificial Intelligence and Tutoring Systems, Los Altos, Ca., 1987

A COMPUTERIZED LEARNING ENVIRONMENT INTEGRATING PRESCRIBED AND FREE STUDENT ACTIVITIES

Luis Osin

*Centre for Educational Technology (C.E.T.)
16 Klausner Street, Ramat Aviv 69011, Israel*

Abstract: Instructional computer programs may be classified, in terms of locus of control, in a spectrum whose extremes are total student control ("learner-directed" or "free"), and total system control ("author-directed" or "prescribed").

The pioneering C.A.I. systems developed in the 60's and 70's were author-directed. These systems included a "teacher model", whose decisions were based on the analysis of a "student model", updated by the computer according to its monitoring of the student performance. The success of this type of system in improving the learning rate in basic skills has been widely documented.

In the 80's, the augmented technological possibilities, and the emphasis in the development of higher order thinking skills, resulted in more emphasis being placed in simulations, games, microworlds and tools. In these programs every student has complete freedom, with tools for exploring the environment s/he is presented with. The lack of evaluations showing a clear student improvement in thinking skills because of the use of these programs is not a reason for abandoning this line of development. The field is too young, and much more has to be learned.

The problem to solve is how to integrate into one computer system, in a coherent form, the reliability of the classical prescribed activities, with the possibilities opened by the free activities.

Our solution to this problem involves a structure of pointers linking the student model, whose knowledge structure is updated according to the computer monitored activities, to a multidimensional taxonomy of free activities.

Instruction and individual differences

The school we are familiar with was conceived in Europe, at the time of the industrial revolution, and was influenced by industrial methods: start from the raw materials (pupils), apply a sequence of standard processes, and the result will be a final product (graduates). What the educational system did not copy from industry refers to responsibility: if the final industrial product fails, the producer tries to change the process; if the educational product fails, the educational system blames the raw material. What the well-intentioned people that tried to design a system for mass education did not realize, is that the raw material is not adequate for the application of one standard process. There is a crucial difference between the industrial and the educational situation: industry may choose its raw materials and establish quality and homogeneity controls before the