Domain Knowledge Structure, Knowledge Representation and Hypotheses Testing

Heinz-Jürgen Thole[1], Claus Möbus[1,2], Olaf Schröder[1]

[1]*OFFIS Institute, Escherweg 2, 26121 Oldenburg, Germany*
[2]*FB Informatik, University, 26111 Oldenburg, Germany*
E-Mail: {Heinz-Juergen.Thole, Claus.Moebus, Olaf.Schroeder}@informatik.uni-oldenburg.de

Abstract: Intelligent problem solving environments (IPSEs) offer students the opportunity to acquire knowledge while working on a sequence of problems chosen from the domain. Up to now we have developed several IPSEs for various curricula and applications (computer science, configuration problems, pneumatics, economic simulation games, causal modelling, and chemistry). On the surface being very different all IPSEs follow a common design theory: the student is encouraged to acquire knowledge by stating and testing hypotheses.

One of these differences is the structure of domain knowledge. The aim of this paper is to show *different realizations* of the hypotheses testing approach for *differently structured domain knowledge*. The domains differ in their knowledge representation, tree-like structure vs. graph-like structure, and their possibility to absolutely locate errors. The interrelations between the domain knowledge, the representation of the diagnostic knowledge, and the contents of the help information are described. First we give a brief overview of our knowledge acquisition theory. Two IPSEs with differently structured knowledge follow. The *diagnostic components* and their relationships to hypothesis testing are discussed and a comparison is made that points at the *commonalities* and the *differences* between the systems that can be traced back to *different domain structures*.

1. Introduction

The long-term success of Computer-Based Training depends on a variety of factors. One of them is the communication of qualitatively outstanding knowledge that is lasting, applicable, transferable, and cost-effective. One approach to achieve this is to enable and encourage free, only weakly constrained problem solving by presenting problem solving tasks from the domain of interest, by enabling the learner to state and test hypotheses, and by offering feedback and explanations, if necessary.

We developed systems with these features in the domains of functional programming, time-dicrete distributed systems, room configurations [8] creation of pneumatic circuits, business simulation games, and modelling and diagnosing in medical domains [9]. We call these systems intelligent problem solving environments (IPSEs). From our point of view IPSEs seem to be the most cost effective intelligent systems for the communication of problem solving knowledge.

In this paper, we want to address the question how differences in domain structure effect the construction of IPSEs. We will introduce a new IPSE, called TAT ("Try and Test"), designed to support the construction of constitutional formulas of reaction equations in the domain of plastics chemistry. We will compare TAT to our IPSE for functional programming, ABSYNT, because in ABSYNT the relevant domain knowledge is *tree-like structured*, wheras in TAT it is *graph-like structured*. In contrast to the domain of constructing structural formulas in chemistry, it is usually not possible to absolutely locate errors in the domain of functional programming. The discussion will show how these *domain differences* affect the *architectures* of the two systems.

The paper is organized as follows: First we give a brief overview of problem solving and knowledge acquisition and its implications for design decisions for IPSEs. Then we will describe two case studies: ABSYNT and TAT. Then we will discuss the commonalities and differences between the systems and show how they can be traced back to structural features of

the domains and their distinction relating to the possibility of error location. We think that this discussion can contribute towards an *epistemology* and domain structure related *taxonomy* of IPSEs.

2. The ISP-DL Knowledge AcquisitionTheory

Our own empirical investigations [10] led to the ISP-DL Theory [11] which is intended to describe processes occurring while problem solving. ISP-DL Theory has three aspects:

• The distinction of *different problem solving phases* [6]. In the *deliberation* phase the problem solver considers several goals and finally chooses one. In the *planning* phase a solution plan is developed to obtain the goal. Then the plan is executed, or *implemented*. Finally the problem solver *evaluates* the result.

• The *impasse driven acquisition of new knowledge* [7, 13, 14]. When knowledge is not sufficient an impasse occurs. In response to an impasse, the problem solver applies weak heuristics, like asking questions and looking for help. Thus the learner obtains new information. As a result of this, the learner may overcome the impasse and acquire new knowledge. Thus impasses trigger the *acquisition* of knowledge.

• The *success driven improvement of existing knowledge*. Successfully used knowledge will be improved so that the number of control decisions and subgoals can be reduced [1, 7].

The ISP-DL Theory motivates the following design principles for IPSEs:

(1) The IPSE should enable *free and unconstrained* problem solving.
(2) The IPSE should not *interrupt* the problem solver but offer information only on demand ("just in time").
(3) The student should have the opportunity to obtain *detailed feedback* and information any time at the *different phases* of problem solving.
(4) The learner should be enabled to make use of her/his *pre-knowledge* as much as possible when asking for help.

3. First Case Study: ABSYNT

ABSYNT ("Abstract Syntax Trees") supports programming novices with help and proposals while they acquire functional programming concepts including recursion. ABSYNT was designed to encourage explorative learning. The ABSYNT system provides a *visual editor* for constructing programs. ABSYNT programs consist of trees built from connected primitive and self-defined operator nodes, parameters, and constants. In addition program plans can be constructed using *goal nodes*. In a *diagnosis-, hypotheses- and help environment* the learner may state the hypothesis that her/his solution proposal (or part of that proposal) to a programming task is correct. The system then analyzes the part of the solution proposal chosen by the student as a hypothesis. As the result, the system gives help and error feedback on the *implementation* and *planning level* by synthesizing complete solutions for the given programming tasks, starting from the learner´s hypothesis. If the hypothesis is embeddable within a complete solution, the learner may ask for completion proposals.

Figure 1 shows a stage of a solution for the programming task "list-reversal". The learner may construct tree representations of mixed terms. Trees represent mixed terms when they contain runnable nodes (rounded operator nodes like "REVERSE" and the parameter "L" in Figure 1) and specification nodes or "goal" nodes (cloud-like shaped nodes like "LIST EMPTY" in Figure 1). These "clouds" represent plans which have to be implemented later by runnable nodes. The learner may state a hypothesis by highlighting (parts of) her/his actual solution proposal. The hypothesis is: "The highlighted part of my solution proposal is embeddable in a correct solution!" The system´s diagnosis component analyzes the hypothesis.

On the first level of feedback, ABSYNT informs the learner whether her/his hypothesis is correct (embeddable). In case of a correct hypothesis, the learner may further ask the system for completion proposals. ABSYNT offers only minimal information, that is, one node at a time for completion in order to stimulate problem solving and self explanation [3]. "How"- and "Why"- explanations are available on demand, too.

One important reason for stating hypothesis about the embeddability of the solution proposals is dependent on the domain. In functional programs, it is usually not possible to absolutely locate errors. Rather, errors consist of inconsistencies between program parts. In ABSYNT hypotheses testing leaves the question what part of a solution proposal to keep and what part to change to the learner in order to avoid additional burden.

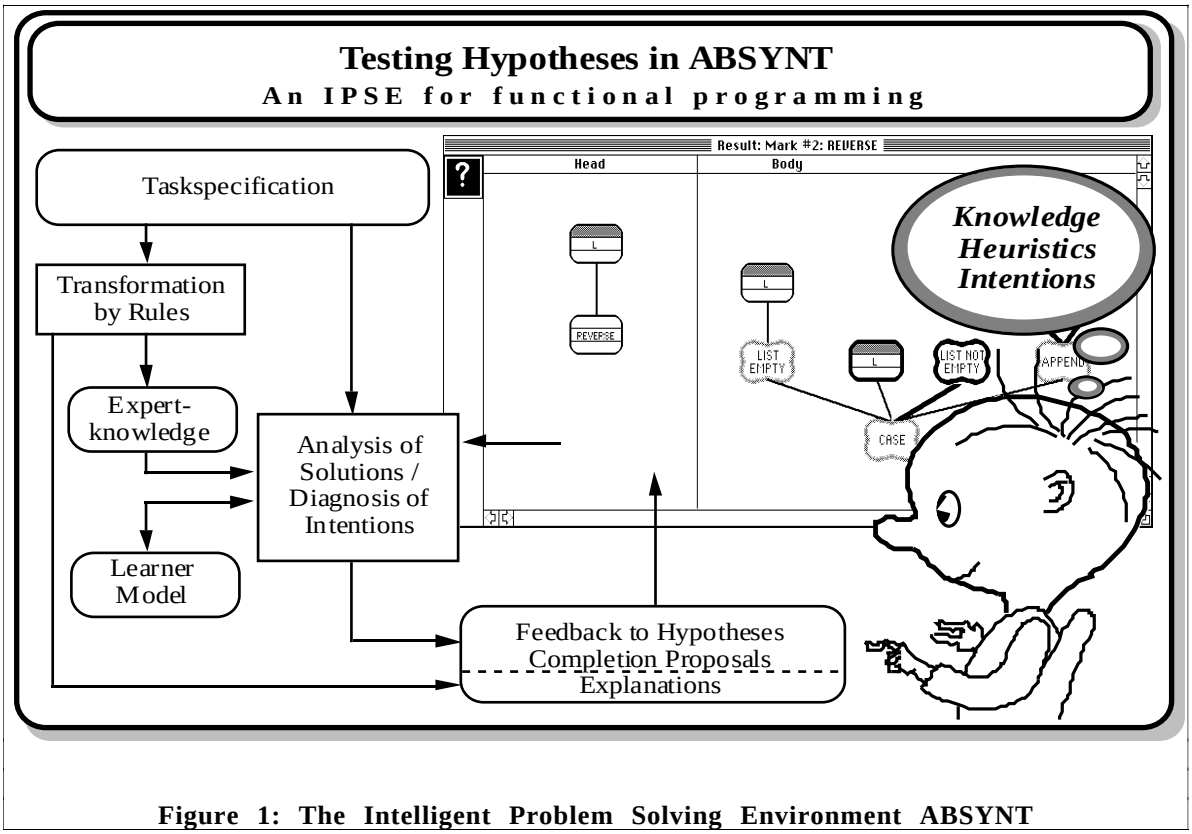

Figure 1: The Intelligent Problem Solving Environment ABSYNT

The *diagnostic knowledge* of ABSYNT consists of a *goals-means-relation* GMR. The GMR consists of about 1100 rules which contains the knowledge about correct solutions for 42 tasks in ABSYNT. With these rules the diagnosis component of ABSYNT is able to parse and generate several millions of solutions. This knowledge can be represented in a *tree structure* which determines the start of the diagnosis process. During diagnosing the trees are handled from the root to the leaves and from left to right (*depth first, left first* strategy). Figure 2 shows examples for GMR 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). 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).

The first rule of Figure 2, 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"

Rule O1 in Figure 2 is an example of a rule implementing one runnable node:

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.

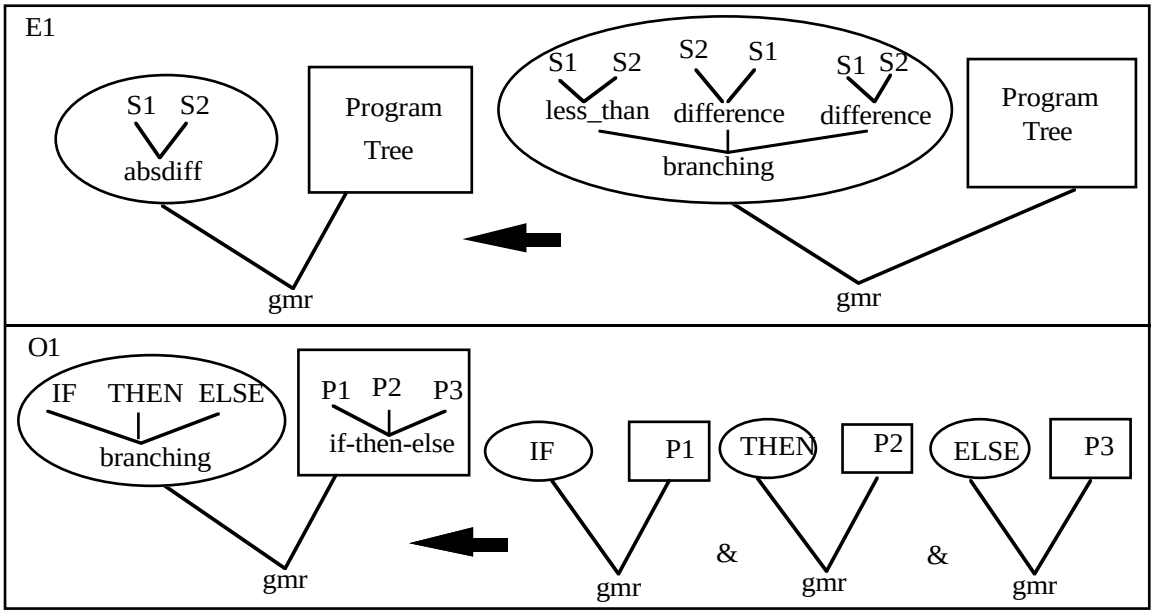

Figure 2:
A goal elaboration rule (E1) and a rule implementing the ABSYNT IF-operator (O1)

In order to analyze the hypotheses of the learners the diagnosis component of ABSYNT is capable not only to parse solution proposals, but also to generate and to complete missing parts of a partial solution. Therefore some control knowledge has to be added to the GMR rules. For example, there has to be some restriction on the number of repeated applications of the commutative law of addition. In ABSYNT the control knowledge is encoded in a meta-interpreter applied to the GMR rules. For efficiency purposes, the knowledge contained in the meta-interpreter is inserted into the GMR rules by a compilation process before runtime.

4. Second Case Study: TAT

TAT ("Try and Test") is an IPSE for training problem solving in the chemistry on plastics and synthetic substances. It is designed as an addendum to a standard book for working on and studying the chemistry of plastics [5] Unlike other systems that support the construction of chemical structural formulas (e.g. ChemLab, [2]), TAT encourages *hypotheses testing* and delivers *knowledged-based help* information on demand. TAT contains 61 tasks concerning the theoretical foundations of polymerization. For getting started, there are some introductory tasks for constructing the chemical structural formulas of some simple synthetic substances, called monomers. In constructing the formulas for these molecules, the learner acquires the concepts of partial charge and ionic charge. These concepts are necessary in order to understand the chemical reaction processes going on in the construction of the molecule chains, called polymers. Now the learner is prepared to construct chemical structural formulas of the reaction equations of the initiating-, growth-, and final reaction processes. TAT contains tasks about different types of polymerization: Radical, anionic, and cationic polymerization, ionic polymerization of heterocycles, addition and condensation polymerization, polymerization by oxidative coupling, and copolymerization. Furthermore the distinct results of stereospecific polymerization are considered.

The learner contructs the structural formulas of the molecules and the reaction equations in a two dimensional graphical editor. In the domain of constructing structural formulas it is *possible to locate errors* absolutely. Therefore, in contrast to ABSYNT, in the system TAT the learner may state hypotheses about the correctness of his solution proposals (and not only about their embeddability). TAT analyzes the hypotheses and gives feedback to the learner on an *increasingly detailed level* in order to favor *self explanation* and *problem solving*. Firstly, the learner is informed whether his hypothesis contains invalid elements or bonds without depicting the errors by detail. TAT gives the feedback: The hypothesis contains wrong elements or bonds (if any). Now the learner may try to find these errors by himself. In the case of missing knowledge the learner may ask TAT for further help. Next, the learner is informed whether the solution contains errors concerning the charge and the lone pairs of electrons. Again, the learner may decide to self explain the information given and return to

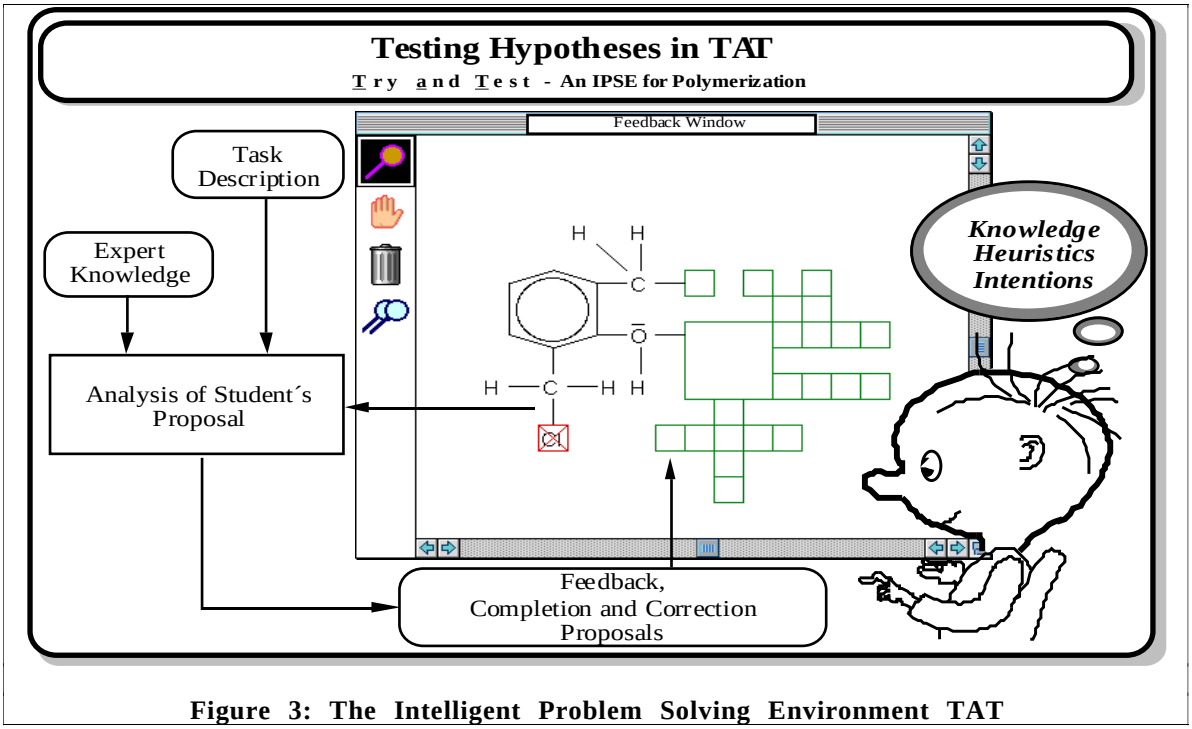

Figure 3: The Intelligent Problem Solving Environment TAT

problem solving, or ask for more information. The last level of feedback consists of four kinds of information:

- Correct parts of the hypothesis are presented in the feedback window.
- The positions of some elements of the hypothesis may have to be changed. These elements are marked by a green square with a diagonal in it.
- Elements of the hypothesis that have to be deleted are marked by a red square with cross.
- Missing elements are indicated by empty green squares.

Figure 3 shows the feedback window to a hypothesis on this level (task: "Polymerization by oxidative coupling of 2,6-dimethylphenole with oxygen"). The left part of the feedback window shows the correct structures of the learner´s hypothesis. The chlorine element on the lower left is wrong, so it is marked by a (red) square with cross. The right part of the feedback window indicates yet missing structure. By clicking onto the empty (green) squares, the learner may uncover the rest of the chemical structural formula step by step.

In contrast to ABSYNT, the *diagnostic knowledge* for analyzing the learner´s structures of the chemical reaction equations is *represented by graphs* rather than by trees. The graphs are coded by lists of atoms and bonds. The diagnostic component of TAT is composed of a bibliography of solution patterns and a set of grammar rules, expressing equivalence transformations [4, 12].

```
task_solution('Vinyl chlorid',
 [atom(1, _, carbon, delta_plus, 0),
  atom(2, _, carbon, delta_minus, 0),
  atom(3, _, hydrogen, delta_plus, 0),
  atom(4, _, hydrogen, delta_plus, 0),
  atom(5, _, hydrogen, delta_plus, 0),
  atom(6, _, chlorine, delta_minus, 0),
  bond(7, _, sigma_pi, 1, 2),
  bond(9, _, sigma, 1, 3),
  bond(10, _, sigma, 1, 4),
  bond(11, _, sigma, 2, 5),
  bond(12, _, sigma, 2, 6)]).
```

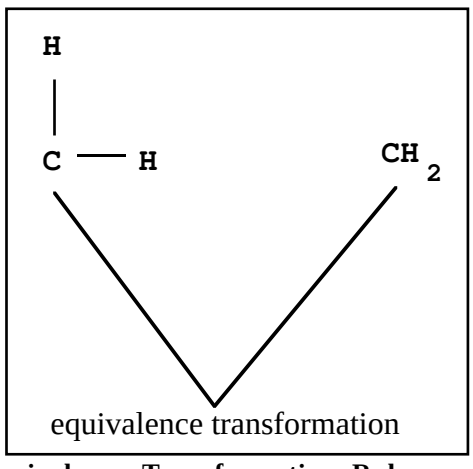

Figure 4: Solution Pattern and Equivalence Transformation Rule

Figure 4 shows a solution pattern list (left hand side) and a graphical representation of an equivalence tranformation rule (right hand side). The atoms are the nodes and the bonds are the links of the graph. The first number in the atom and bond terms is simply an instance number. The "0" in the atom terms indicates the number of lone electron pairs, relevant for the polymerization reactions according to A. FRANCK (Franck, 1996). The last two numbers in the bond terms indicate the atoms linked by the bond, "sigma_pi" is a double bond, "sigma" is a single bond. The example of a transformation rule expresses the equivalence between the structure (one carbon atom, two hydrogens, two sigma bonds) and the group symbol (CH_2).

When TAT analyzes a learner´s hypothesis, the atoms and the bonds of the hypothesis are matched to the atoms and bonds of the stored solution representation(s) like the one just shown. The order of processing is *not predefined*. For example, any C-atom of the learner´s hypothesis may be matched to any C-atom of the system´s solution list. TAT creates *all possible matchings* of the polyvalent atoms between the hypothesis and the solution representation until it detects that the hypothesis is embeddable in one of the solution representations for the given task, or until it has found that embedding that leads to the *minimal deviation* between the hypothesis and the solution representation. This graph matching process has to take care of the fact that the position of bonds is not determined in most cases, except for stereoisomeres like malein acid (cis-form) and fumar acid (trans-form). For example, if the chlorine atom and the upper right hydrogen atom in the rightmost structure of Figure 5 switch their position, the graphical formula is still correct.

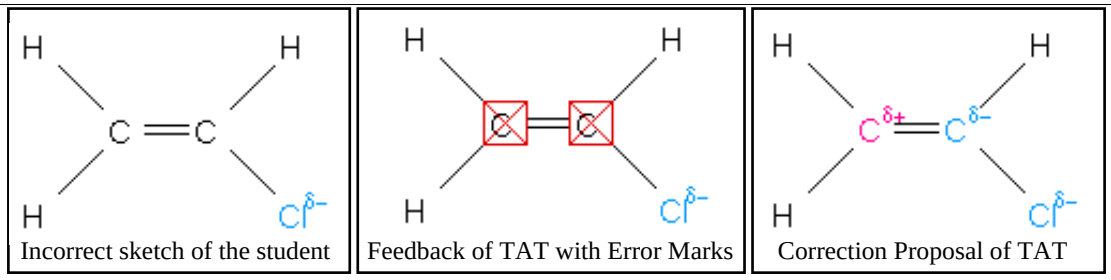

Figure 5: Hypothesis to vinyl chloride with feedback of TAT on most detailed level

In aromatic hydrocarbons, the bonds at benzene rings need special handling. There are ortho-, meta- and para-positions of substituents. These different positions have to be distinguished (Figure 6, methyl group "CH_3", any organic group "R"). In ortho-position, two substituents are "neighbors". In meta-position, there is one C-atom with unsubstituted hydrogen atom (or another different substituent) in between. In para-position, the substituents are located in opposite position. Each position can be constructed in several ways. For example, there are six graphical structures for the para-position at a benzene ring. In TAT, substituents at benzene rings are consecutively numbered, starting from an arbitrary point. In the stored graph representations of the solutions this starting point is represented as a variable, so each possibility can be matched.

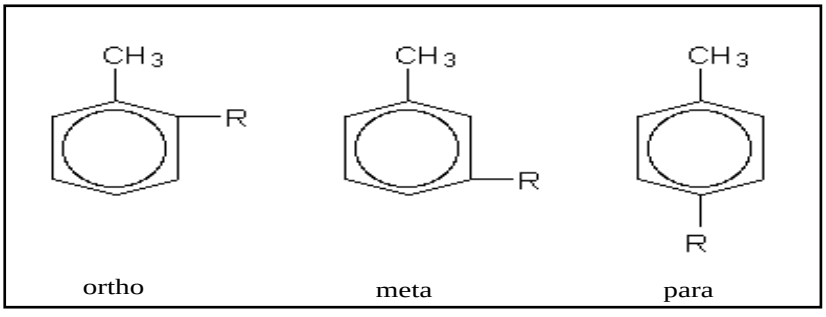

Figure 6: Benzene rings with ortho- meta- und para-positions of substituted groups

5. Comparison of the Diagnostic Processes of ABSYNT and TAT

In this section, we want to point out the *commonalities* and *differences* of diagnostic processes for two domains differing in two respects: (i) the domain structure is tree structured vs. graph

structured, (ii) errors cannot vs. can be absolutely localized. In ABSYNT, functional programming, we have tree structures without absolute error location. In TAT, constructing graphical formulas in chemistry, we have graph structures with absolute error location. Table 1 points out these differences.

Table 1: Differences in task domains between ABSYNT and TAT

	ABSYNT	TAT
number of solutions for a task	high	low
number of ways to graphically represent one solution	low	high
representation of solutions	tree-like	graph-like
possibility of general error location	not given	given
kind of error detection	inconsistency errors	"absolute" errors

In ABSYNT, even under simple restrictions there are often several millions of solutions to one task. This huge solution space is based on different planning steps. The aims of ABSYNT are to adaptively support any problem solving phase on any grainsize of help information and to put explanations at the leaners disposal. This is the reason why the diagnostic component of ABSYNT is developed as a derivation system consisting of rules containing only one planning step or one implementation step. In TAT, there are only a few ways (sometimes even only one way) to express the structural formulas of the chemical reaction processes. The aims of TAT are to locate errors in structure formulas and to give completion proposals on the "atom & bond"-level. Solution cases and equivalence rules are appropriate for these tasks.

In ABSYNT, every solution is topologically unique, with the exception of the commutative laws of some operations like addition. In TAT, we may rotate the graphical representation of a structural formula in many ways without changing it semantically, with the expection of stereoisomerism and substituents of benzene rings. Functional programs are tree-like, chemical structural formulas are graph-like. Finally, as stated, in functional programming we have *inconsistency errors* (parts of a program do not "fit together"). The learner may choose the part of his non-embeddable hypothesis to maintain by himself. In TAT is possible to determine the error locations (if any), so we might have *"absolute" errors*. For example, vinyl chloride must not contain a bromide atom. These kinds of hard restrictions do not hold for functional programs (A solution for "list-reverse" is not necessary wrong just because it contains an addition operator). On demand ABSYNT offers "why"- and "how"-explanations generated from the GMR rules. TAT is not able to explain the detected errors or the completion proposals. These *differences of task domains* lead to *differences in the diagnosis process* which are shown in Table 2.

Table 2: Differences in diagnosis between ABSYNT and TAT

	ABSYNT	TAT
representation of domain knowledge	rules defining a goals-means-relation	stored solution patterns and equivalence transformation rules
diagnosis process	application of goals-means-relation to actual solution	fitting stored solution patterns to actual graphical orientation
diagnosis results	statement about embeddability of hypothesis	statement about correctness of hypothesis
	If true: Complete solution containing	If true: Complete solution containing

	ABSYNT	TAT
	the hypothesis If false: The learner may choose the part of his hypothesis to maintain	the hypothesis If false: Minimal list of errors
explanations	"Why"- and "How"- explanations	not available

6. Summary

This paper has shown again that the design of IPSEs [8, 9] including the hypotheses testing approach is feasable for very diverse domains. Mainly the comparison shows that every domain needs special care to some changes. In the first place, these necessary changes and extensions depend on the special aims of the intelligent problem solving environment. Some of these requirements may depend on features of the domain (e.g. the error location), others are domain independent (e. g. supporting the planning phase). So, the knowledge acquisition process and the specification of the system´s capabilities have to be done precisely with a high amount of accuracy to be able to choose the appropriate knowledge representation.

References

[1] Anderson, J.R. (1993). Rules of the Mind. Hillsdale: Erlbaum

[2] Byrne, M. et al. (1996). The Role of Student Tasks in Accessing Cognitive Media Types. In D. C. Edelson & E. A. Domeshek (eds), International Conference on the Learning Sciences, 1996, Proceedings of ICLS 96, Northwestern University, Evanston, IL USA, July 25-27, 1996, Charlottesville: AACE, 114-119

[3] Chi, M.T., De Leeuw, N., Chiu, M.-H. & LaVancher, Chr. (1994). Eliciting Self-Explanations Improves Understanding, Cognitive Science, 18, 439-477

[4] Conrad, M. (1995). Anwendung algebraischer Graphtransformationen auf ausgewählte Probleme der Wissensrepräsentation und -verarbeitung, TU-Berlin, Forschungsbericht des Fachbereichs Informatik, Bericht-Nr. 95-11

[5] Franck, A. (1996). Kunststoff-Kompendium, Vogel-Verlag

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

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

[8] Möbus, C. (1995). Towards an Epistemology of Intelligent Problem Solving Environments: The Hypothesis Testing Approach, in J. Greer (ed), Artificial Intelligence in Education, Proceedings of AI-ED 95, Washington, D.C., August 16-19, 1995, Charlottesville: AACE, 138-145

[9] Möbus, C. (1996). Towards an Epistemology of Intelligent Design and Modelling Environments: The Hypothesis Testing Approach, in P. Brna, A. Paiva, J. Self (eds), Euro AI-ed, European Conference on Artificial Intelligence in Education, Proceedings of EuroAIED, Lisbon, Portugal, September 30 - October 2, 1996, 52 - 58

[10] Möbus, C., Schröder, O. (1993). The Acquisition of Functional Planning- and Programming Knowledge: Diagnosis, Modelling, and User-Adapted Help, in G. Strube, K.F. Wender (eds), The Cognitive Psychology of Knowledge. The German Wissenspsychologie Project, Amsterdam: Elsevier (North-Holland), 233-261

[11] Möbus, C., Schröder, O. & Thole, H.J. (1994). Diagnosing and Evaluating the Acquisition Process of Programming Schemata, in J.E. Greer, G. McCalla (eds), Student Modelling: The Key to Individualized Knowledge-Based Instruction (Proceedings of the NATO Advanced Research Workshop on Student Modelling, in St.Adele, Quebec, Canada), Berlin: Springer (NATO ASI Series F: Computer and Systems Sciences, Vol. 125), 211-264

[12] Nagl, M. (1979]. Graph-Grammatiken, Braunschweig, Vieweg

[13] Newell, A. (1990). Unified Theories of Cognition, Cambridge, Mass.: Harvard University Press

[14] VanLehn, K. (1988). Toward a Theory of Impasse-Driven Learning, in H. Mandl, A. Lesgold (eds), Learning Issues for Intelligent Tutoring Systems, Berlin: Springer, 19-41

Acknowledgements: We want to express our thanks to Jörg Folckers for implementing the interface of ABSYNT and to Gerhard Grieb for assisting the knowledge acquisition process for TAT. Last but not least we want to thank Thomas Herlyn for carrying out essential work in the completion of the product TAT available on PC.