Artificial Intelligence in Education, 1993

edited by
Paul Brna
Stellan Ohlsson
Helen Pain

Proceedings of *AI-ED 93*

World Conference on
Artificial Intelligence in Education

Edinburgh, Scotland; 23-27 August 1993

AACE Association for the Advancement of Computing in Education

Artificial Intelligence in Education, 1993

edited by
Paul Brna
Stellan Ohlsson
Helen Pain

Proceedings of *AI-ED 93*
World Conference on
Artificial Intelligence in Education
Edinburgh, Scotland; 23-27 August 1993

AACE Association for the Advancement of Computing in Education

Published by

Association for the Advancement of Computing in Education (AACE)
P.O. Box 2966
Charlottesville, VA 22902 USA

Printed in the USA

ISBN 1-880094-08-8

Designing Help for Viewpoint Centered Planning of Petri Nets

O. SCHRÖDER, C. MÖBUS, K. PITSCHKE
Department of Computational Science
Oldenburg University, 26111 Oldenburg, Germany
E-Mail: Claus.Moebus@arbi.informatik.uni-oldenburg.de

Abstract: PETRI-HELP supports novices in learning to model with condition-event Petri nets. It is based on a theoretical framework recommending that a help system should *offer* help, let the learner use *pre-knowledge*, and support different problem solving *levels*. In PETRI-HELP the learner creates Petri nets for given tasks, test hypotheses about the solutions or fragments of them, and receive feedback, completions, and correction proposals.
This help refers to Petri net places, transitions, and links. It does not yet support *planning*, i.e., creating an abstract solution idea and postponing implementation decisions, as recommended by the theory and by our empirical work. This paper describes an approach to support planning within PETRI-HELP. It is based on *transformation* rules (Olderog, 1991) for the systematical derivation of a Petri net from a specification composed of viewpoints of different agents. Applying this approach to PETRI-HELP will enable the learner to propose solution ideas, test hypotheses, and get help on an abstract "goal level".

Introduction

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

In order to support design decisions for the development of an intelligent help system, a theoretical framework of problem solving and learning is needed. Our ISP-DL Theory (impasse - success - problem solving - driven learning theory) attempts to integrate impasse-driven learning (Laird, Rosenbloom & Newell, 1986; 1987; Newell, 1990; Rosenbloom et al., 1991; van Lehn, 1988; 1991), success-driven learning (e.g., Anderson, 1983; 1986; 1989; Wolff, 1987; 1991), and phases of problem solving (Gollwitzer, 1990). Briefly, the ISP-DL Theory (Möbus, Schröder & Thole, 1991; 1992) states that problem solving processes may consist of four phases: The problem solver (PS) *deliberates* with the result of choosing a goal to persue; then a *plan* to reach the goal is created, the plan is *executed*, and the obtained result is *evaluated*. *Impasses* might result at several points in this process: The PS might not be able to choose a goal, or the plan cannot be created, or execution is not possible, or the obtained result is not satisfying. The PS reacts to an impasse by problem solving, using weak *heuristics* like looking for help, asking, cheating. As a result, the PS may overcome the impasse and acquire new knowledge (impasse-driven learning). But alternatively, the information obtained may not be helpful but confusing, so the learner might encounter a secondary impasse (Brown & van Lehn 1980). Finally, if a problem has been successfully solved without impasses, then the knowledge used is optimized (success-driven learning).

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

- According to the theory, the learner will look for and appreciate help if he or she is caught in an impasse. Without an impasse there is no need for help. So the system should not interrupt the learner (see for example also Winkels & Breuker, 1990), but *offer* help.
- The learner should be prevented from trapping into secondary impasses which may lead away from the original problem solving. So *pre-knowledge* should be useable at impasses as much as possible. One

way to realize this principle is to let the learner test hypotheses about her or his solutions, and get help and proposals from the system. This leaves the activity on the learner's side, the learner is not disturbed by unwanted system comments. Secondly, accounting for pre-knowledge means that help should be adapted to the knowledge state of the learner. Help should be *user-oriented.* This requires a learner model.

• According to the ISP-DL Theory, help should be provided at different phases of problem solving because impasses may arise in all phases. So a help system should support *deliberating*, *planning*, *executing*, and *evaluating* solution proposals. Help should be *problem phase oriented.*

Centered around the ISP-DL Theory, we develop two help systems: ABSYNT supports functional programming in a visual language (Möbus, Schröder & Thole, 1992), and PETRI-HELP supports modelling with condition-event Petri nets (Möbus, Pitschke & Schröder, 1992). According to the theory, modelling with Petri nets is a problem solving activity consisting of the following sub-activities:

- to develop specifications of systems or processes to be modelled ("*deliberating*")
- to plan a Petri net solution for a given specification ("*planning*")
- to actually construct a Petri net ("*executing*")
- to evaluate the resulting net, for example whether it meets the specification ("*evaluating*").

So the skill of modelling with Petri nets may be decomposed into four corresponding subskills to be learned by a Petri net modeller. Consequently, PETRI-HELP should support the acquisition of these four subskills. Currently, mainly "execution" and "evaluation" are supported, as will be shown in the next section which gives a short overview of the current implementation state of PETRI-HELP, and of some empirical results. The main section of this paper is concerned with an approach to support the learner's *planning* processes while constructing Petri nets for given tasks. In the final concluding section, we will also touch upon the issue how to support the learner's *deliberation* processes.

A Brief Overview of PETRI-HELP

PETRI-HELP (Möbus, Pitschke & Schröder, 1992) is intended to support novices while constructing condition-event Petri nets. The learner may create Petri net solutions to given tasks which are specified as sets of temporal logic formulas. The learner may also hypothesize which (sub)set of the formulas constituting a task description he thinks is fulfilled by his current solution proposal. The system then gives feedback about fulfilled and unfulfilled formulas, and may deliver proposals how to complete or to correct the actual state of the solution.

There are two reasons for specifying the tasks in PETRI-HELP as sets of temporal logic formulas. Firstly, well-defined tasks are necessary in general in order for the system to be able to evaluate Petri net solution proposals. Secondly, temporal logic specifications allow to verify Petri net proposals by model checking (Clarke, Emerson & Sistla, 1986; Josko, 1990). So the learner's task is to construct a condition-event Petri net that satisfies the given set of temporal logic formulas (i.e., a given task description). Due to model checking, the system supports free, unconstrained problem solving in that *any* hypothesis of the learner can be evaluated, and feedback and completions can be given to it based on rules learned by the system.

Figure 1 shows the temporal-logic specification and an empirical solution proposal to a task, "Bavarian Biergarten". [], ◊, O are the temporal logic operators. Informally, [] means "always" ("it is always true that ..."), ◊ means "eventually" ("now or at some point in future it will be true that ..."), and O means "nexttime ("at the next point in time it will be true that ..."). So for example [] (Ws --> ◊Wro) means: "It is always true that if the waiter sleeps then he will eventually be ready to accept an order." In the condition-event Petri net, circles represent places (conditions, states), and rectangles represent transitions (events). The condition represented by a place is true if the place contains a token.

PETRI-HELP consists of the following components:

- an *editor* for constructing condition-event Petri nets and for simulating them.
- a *task* window where the actual task (= set of temporal logic formulas) is presented. Currently there are 10 tasks in PETRI-HELP, for example, modelling events in a restaurant (Figure 1), in a library, the use of a telephone, and processes in natural (photosynthesis) as well as technical systems.
- a *hypotheses window* where the learner may hypothesize which (sub)set of the task formulas he or she considers fulfilled by the actual Petri net proposal.
- a *feedback window* where the learner is informed about which formulas of his hypothesis are fulfilled and which ones not. This analysis is based on a case graph constructor and a model checking algorithm.

Ws : Waiter is sleeping	[] (Ws → ◊ Wro)	[] (¬(Ws ∧ Wro))
Wro : Waiter is ready to accept order	[] (R ∧ Ws → ◊ Wrs)	[] (¬(Ws ∧Wrs))
Wrs : Waiter is ready to serve	[] (Wro → ◊ (Ws ∧ K))	[] (¬(Wro ∧ Wrs))
K : Kitchen got order	[] (K → ◊ P)	[] (Ws v Wro v Wrs)
P : Preparation of the meal	[] (R ∧ Wro → ◊ Wrs)	[] (Wrs → ◊ Ws)
R : Meal is ready		[] (P → ◊ R)

O means "nexttime", ◊ means "eventually", [] means "always"

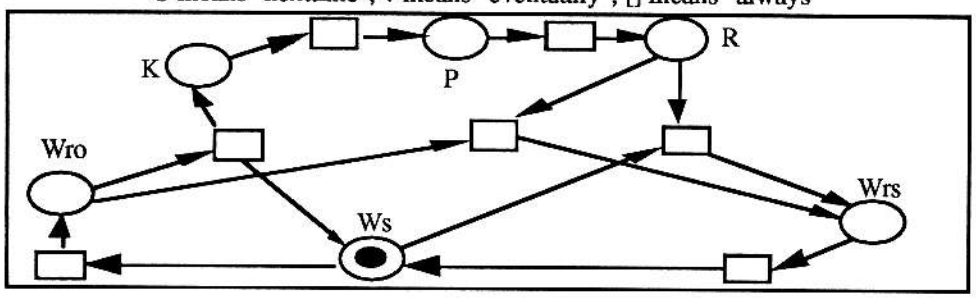

Figure 1. Specification and solution to the task "Bavarian biergarten"

• a *completion proposal window* where the learner gets hints about how to continue with the Petri net proposal. The completion / correction proposal tells the student which states, transitions, and links between them to add and which ones to delete, if any. The completion proposal is based on rules learned by the system from prior sessions with other students. These rules associate subsets of the task formulas to Petri net fragments fulfilling them.

So PETRI-HELP currently supplies two kinds of *help*: Feedback of fulfilled and unfulfilled formulas, and completions / corrections of Petri net proposals. How is PETRI-HELP related to the ISP-DL Theory?

• PETRI-HELP *offers* information. It is up to the learner to state hypotheses (subsets of the set of task formulas), and to ask for completion proposals. The activity is on the learner's side.
• PETRI-HELP lets the learner use his or her pre-knowledge because the learner tests *hypotheses*, gets completion proposals, and decides what to keep and what to change. In addition, the learner always gets only *minimal* information to resolve the actual impasse without telling too much about the solution.
• Yet missing in PETRI-HELP is a user-centered proposal of solution completions. This requires a model of the individual learner (Möbus, Schröder & Thole, 1992) not yet realized in this domain.
• Also missing is help for a "planning phase": information which is more abstract than the actual Petri net constructs (states, transitions, and links). Currently, PETRI-HELP provides help for two different problem solving phases: the execution level (or "language level", the level of Petri net constructs), and the evaluation level (hypotheses testing, net simulation). But the learner gets no help aimed at impasses which could arise and be met at a more abstract level than Petri net states, transitions, and links. Furthermore, there is no support of a "deliberation" phase where the learner may want to find out what kind of task he or she wants to work on in the first place.

Some of the current shortcomings of PETRI-HELP are mirrored in the results of our empirical work. 25 students working with PETRI-HELP appreciated the two kinds of help differently. Testing hypotheses was accepted and widely used. Students reported that receiving feedback of fulfilled and unfulfilled fomulas gave information about where to proceed without obstructing problem solving. In contrast, the reactions to the completions and correction proposals were less positive. For example, some students criticized that this kind of help was too detailed and took part of the problem solving away from them. In addition, places, transitions, and links were just presented but not explained.

So the ISP-DL Theory and empirical studies led to a common implication for PETRI-HELP: Help information should be provided on a level more abstract than places, transitions, and links. This information should support planning and hypotheses testing of plans at early stages of problem solving, and it should support their stepwise transformation into more detailed ones.

Incorporating Viewpoint Centered Planning into PETRI-HELP

We investigate an approach developed by Olderog (1991) that allows to transform a task description (a specification) into a condition-event Petri net by making use of intermediate specifications as well as "mixed terms" (terms composed of specification fragments and Petri net fragments). Basically, the intention of Olderog is to derive a description of the operational behavior of concurrent processes from a set of logical formulas specifying these processes. A transformation begins with a set of formulas and constructs a process term from it by transformation rules. The process term expresses a possibly concurrent process in an abstract programming language. A Petri net, with an explicit representation of concurrency, defines the semantics of the process term. It can be derived from the term by net construction rules. So a derivation chain can be constructed from the specification to the Petri net.

Viewpoint Centered Specification

The specification of a process is stated in trace logic (i.e., Hoare, 1985). As a simple example, Figure 2 shows a possible trace-logic specification of (a variant of) the "Bavarian Biergarten":

When observing the events in the "Bavarian Biergarten", we may notice a lot of things. At the beginning, the waiter may be sleeping (Ws). Then after some other things not of concern here (denoted by @), the waiter is ready to accept an order (Wro) from a guest. Next, the kitchen receives the order from the waiter (K). (Of course, there are events in between: The waiter approaches the table, welcomes the guest, receives his order and goes to the kitchen. But these steps are ignored here.) Then the kitchen prepares the meal (P). When the meal is ready (R), the waiter is ready to serve (Wrs), and so on.

Trace of events in the "Bavarian Biergarten":
Ws. @. @. Wro. K. P. @. R. Wrs. Ws. @. Wro. K. @. P ...

Trace logic specification of the "Bavarian Biergarten":
trace $\downarrow$ {K, P, R} $\in$ pref(K.P.R)* Λ
trace $\downarrow$ {Ws, Wro, K, R, Wrs} $\in$ pref(Ws.Wro.K.R.Wrs)*

View of kitchen: K.P.R.K.P.R....
View of waiter: Ws.Wro.K.R.Wrs.Ws.Wro.K.R.Wrs...

Figure 2. Trace and trace logic specification of the "Bavarian Biergarten"

We may look at this chain of events from different perspectives (see Figure 2). From the kitchen´s viewpoint, there are only three relevant events: K, P, R. This is expresssed by "trace $\downarrow$ {K, P, R} $\in$ pref(K.P.R)*". This means: If the trace (= the observed chain of events) is filtered ($\downarrow$) by looking only at the events K, P, R, then a trace is obtained that is an element of the set "pref(K.P.R)*", which is {ε, K, K.P, K.P.R, K.P.R.K, K.P.R.K.P, K.P.R.K.P.R, ...} (ε is the empty trace). So the kitchen´s view is a succession of events K.P.R.K.P.R.K... Similarly, we can take the waiter´s point of view. It is a succession of sleeping (Ws), taking orders (Wro), passing them to the kitchen (K), being told that a meal is ready (R), and serving it (Wrs).

Viewpoint Centered Planning and Implementation

In order to make the transformational approach of Olderog (1991) useable within PETRI-HELP, several simplifications were made. Figure 3 shows our graphical representations of some transformation rules. In these representations, each rule has three parts. The upper part contains the name of the rule, the middle part may contain conditions, and the lower part contains a statement. The parallelism rule has no condition. It says that a specification which is expressed as a conjunction of terms S and T (S Λ T) is equivalent to two specifications S and T which have to be implemented as parallel nets. On the PETRI-HELP screen, we may create two *goal regions* labeled by S and T, which have yet to be implemented by Petri net fragments. Thus a goal region represents a specification of a task or subtask: the goal to create a Petri net fragment that is equivalent to that specification. The dotted crossing lines between the goal regions in Figure 3 mean that these nets will have to be

synchronized (which is specified by the net combination rule). For example, applying the parallelism rule to the trace specification of the Bavarian Biergarten in Figure 2 leads to the goal regions shown in Figure 4.

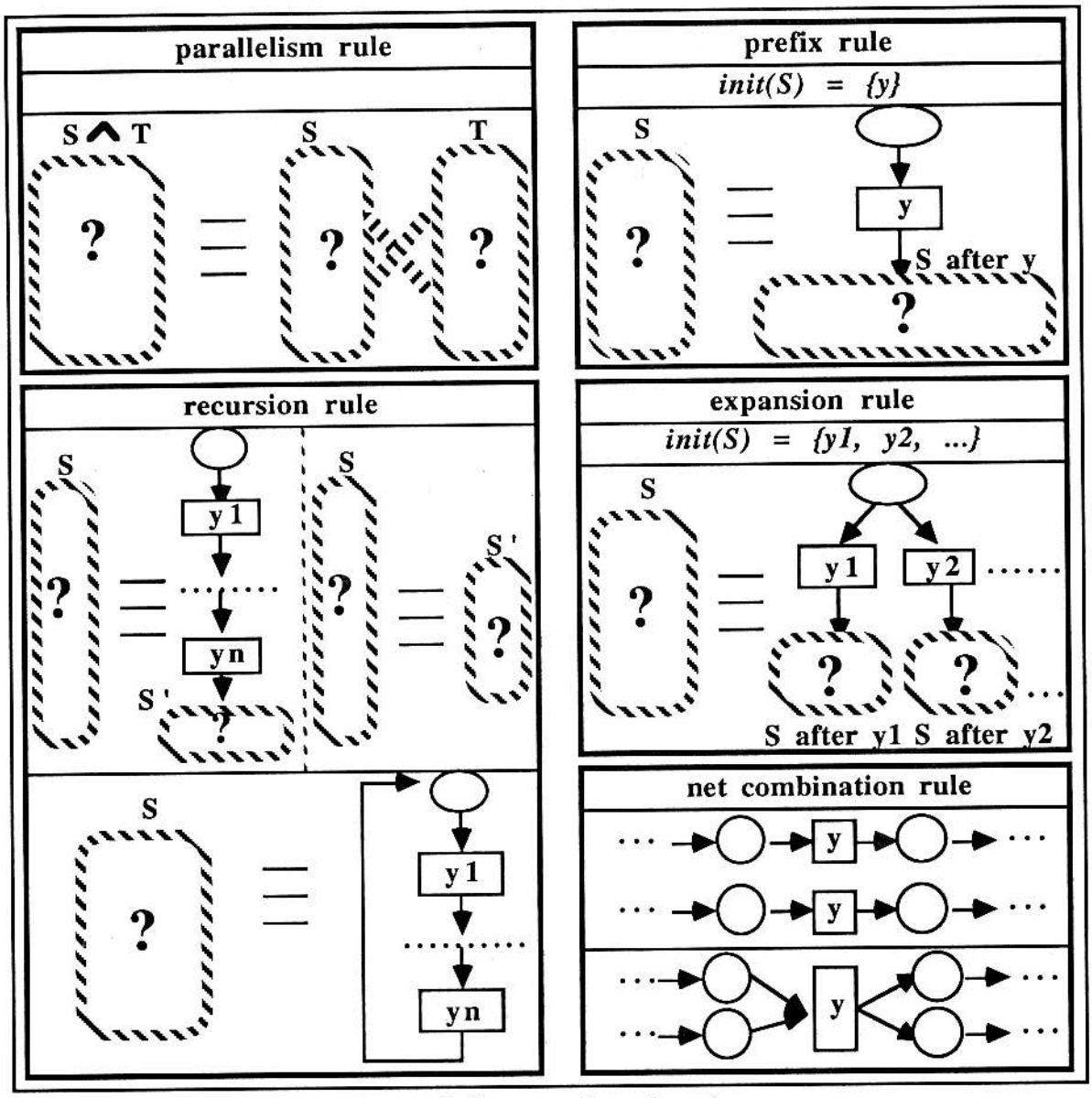

Figure 3. Some transformation rules

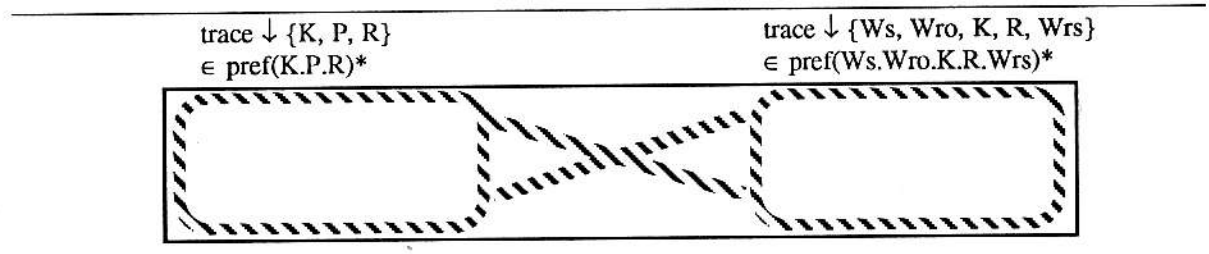

Figure 4. Applying the parallelism rule to the trace specification in Figure 2

Now we can construct the two nets for the kitchen and for the waiter separately, and finally we synchronize them. We will show the construction of the "kitchen net", the "waiter net" is constructed similarly:

"init(S)" is the set of next possible events of a process specified by S. This set might be empty (deadlock, not shown), contain one element (then the prefix rule is applicable), or more than one element (handled by the expansion rule). "trace ↓ {K, P, R} ∈ pref(K.P.R)*" (goal region 5a in Figure 5) has one possible next element: K. (Since all non-empty traces in this set begin with K, any value of the variable "trace" must start with K in order for the formula to be true.) So the prefix rule (Figure 3) is applicable. Now we can generate a place leading to a transition labeled with K leading to a goal region again (Figure 5b). The place gets a token because it is the starting place in this construction. The new goal region represents the kitchen *after* having received an order from the waiter: "K.trace ↓ {K, P, R} ∈ pref(K.P.R)*". (If "K.trace", which is any trace starting with K, is projected onto {K, P, R}, then the result is in the set "pref(K.P.R)*".) Figure 5b represents a mixed expression because it is a mixture of specification parts and places, transitions, and links. Next, the prefix rule is applicable again because the expression "K.trace ↓ {K, P, R} ∈ pref(K.P.R)*" can only be true if any value of the variable "trace" begins with P. (Figure 5c). After three applications of the prefix rule (Figure 5d), the expression "K.P.R.trace ↓ {K, P, R} ∈ pref(K.P.R)*" will be obtained. This expression is equivalent to the original expression "trace ↓ {K, P, R} ∈ pref(K.P.R)*", because if a trace is in the set "pref(K.P.R)*", then the same trace preceded by K.P.R will also be in this set, and vice versa. Thus the recursion rule is applicable (with S substituted by "trace ↓ {K, P, R} ∈ pref(K.P.R)*", and S' substituted by "K.P.R.trace ↓ {K, P, R} ∈ pref(K.P.R)*"). The recursion rule states that if a specification S is equivalent to a mixed expression containing a specification S', and S and S' are equivalent as well, then changing that mixed expression by removing S' and introducing recursion still keeps it equivalent to S. Figure 5e shows the result of its application.

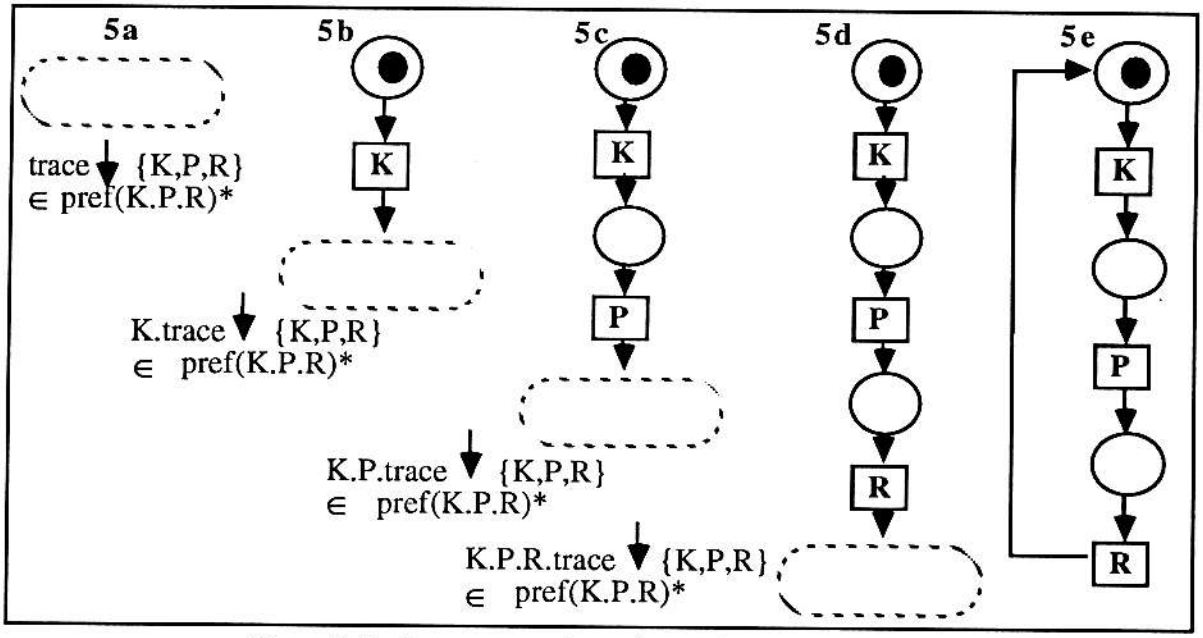

Figure 5. Petri net construction, using prefix and recursion rule

Viewpoint Centered Synchronization

When nets have been created for different viewpoints, they have to be glued together to one single net. This is achieved by the net combination rule (Figure 3). It says that if two transitions with the same name occur in two nets, then an arrow should lead from each of its preconditions (in both nets) to the transition, and an arrow should lead from the transition to each postcondition as well. The net combination rule will combine nets for the two views in Figure 2 to the net shown in Figure 6b. (K and R are the synchronizing transitions.)

Using the transformation approach, different *strategies* are possible. For example, we can take a look at the two different views of Figure 2 *simultaneously* and thus avoid the parallelism rule. The result is shown in Figure 6a. Alternatively, the components of a net can be developed separately by using the parallelism rule, and then glued together by the net combination rule, as shown in Figure 6b. In general, intermediate strategies between maximal sequentiality and maximal parallelism are possible too, though not in this simple example.

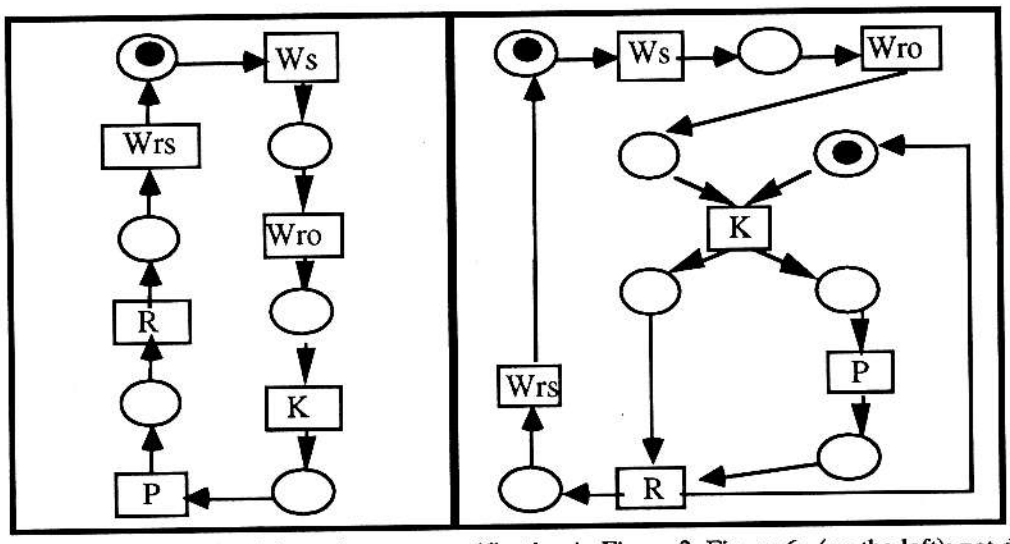

Figure 6. Two Petri nets derived from the trace specification in Figure 2. Figure 6a (on the left): net derived by a sequential strategy, Figure 6b (on the right): net derived by a parallel strategy

Supporting planning of Petri nets in PETRI-HELP

How can the transformational approach support learners in constructing Petri nets?

1. If the learner does not know how to proceed and asks for a completion proposal, the system may *offer goal regions* (as illustrated in Figures 4 and 5) as help. So the learner would be provided with descriptions of subtasks yet to be solved, and with recommendations how to decompose the task into subtasks. Thus the system would not be restricted to help on the level of places, transitions, and links.

2. The learner may state *hypotheses about goal regions*, not only about Petri net fragments. So the learner may get information whether he or she is "on the right track" at very early planning stages. For example, the learner may ask the system if it is appropriate to structure the problem of Figure 2 into two parallel components (Figure 4) without bothering about what the components will exactly look like. So the learner may postpone implementation considerations and work with partial plans and mixed expressions.

3. The learner may receive direct guidance in Petri net construction by using the *transformation rules as help*. So the learner may create a Petri net solution by stepwise application of the rules, that is, in a systematical, derivational way. In order to find out whether this is feasable, we carried out a single subject study. The subject was a novice concerning Petri nets. Her task was to create the "restaurant" net with paper and pencil, using graphical representations of the transformation rules as shown in Figure 3. The subject adopted a maximally parallel strategy. She needed some assistance for applying the parallelism and recursion rule, she did not immediately realize their applicability. But in general, she had no serious problems with this task. This preliminary result suggests that the approach is feasable as a basis for supporting novices in Petri net design.

Conclusions

In PETRI-HELP as currently implemented, tasks are specified as temporal logic formulas. One of their advantages is that there are no restrictions to possible solutions (Petri nets), except that the set of formulas has to be fulfilled. Any solution proposal can be analyzed. So the temporal logic approach allows "free", unconstrained problem solving, but does not support planning "above" the level of places, transitions, and links, and it does not support a systematic, guided construction of a solution.

We have shown an approach to incorporate planning into Petri net design which is more abstract than the Petri net constructs. The approach allows to systematically construct a net proposal, starting from a trace logic formula. Furthermore, it is a sound basis for letting the learner express initial ideas, partial plans, test hypotheses about them, and receive proposals from the system at the same level. The learner is enabled to think about specifications (and "mixed terms") without bothering about their implementation from the beginning.

Our future work is concerned with combining the temporal logic and the trace logic approach so that free problem solving, guided systematical problem solving, and abstract planning are possible as well.

With respect to our theoretical framework, the problem solving level of *deliberation* remains still uncovered. This means that the learner should be supported by PETRI-HELP in *generating* the specification of some system or process. Then the Petri net solution created by the learner would be checked against this specification. In assisting the learner to create a specification, the system may help the learner and help in a dialog to acquire and to integrate the knowledge needed.

References

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

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

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

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

Clarke, E.M., Emerson, F.A. & Sistla, A.P. (1986). Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. on Programming Languages and Systems, 8(2),* 244-263.

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.

Hoare, C.A.R. (1985). *Communicating sequential processes.* Englewood Cliffs: Prentice Hall.

Josko, B. (1990). Verifying the correctness of AADL modules using model checking. In J.W. de Bakker, W.P. de Roever & G. Rozenberg (eds): *Proceedings REX-Workshop on stepwise refinement of distributed systems: models, formalisms, correctness.* Berlin: Springer, LNCS 430, 387-400.

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

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

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

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

Möbus, C., Schröder, O. & Thole, H.-J. (1992). A model of the acquisition and improvement of domain knowledge for functional programming. *Journal of Artificial Intelligence in Education, 3(4),* 449-476.

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

Olderog, E.-R. (1991). *Nets, terms, and formulas.* Cambridge: Cambridge University Press.

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

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

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

Winkels, R. & Breuker, J. (1990). Discourse planning in intelligent help systems. In C. Frasson & G. Gauthier (eds): *Intelligent tutoring systems.* Norwood: Ablex, 124-139.

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

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

Acknowledgements

We thank J. FOLCKERS for implementing the PETRI-HELP interface.
This research was supported by the Stiftung Volkswagenwerk (Az. 210-70631/9-13-14/89)