Research report 113

LEARNING TO PROGRAM: A COGNITIVE MODEL FOR AN ITS

Iain D Craig & Robert F Thomas

(RR113)

Abstract

Programming is a complex skill acquired only after considerable practice. This paper outlines a hierarchical cognitive model of problem solving and programming. The model accounts for the way students acquire programming knowledge and forms a basis for an integrated problem solving and PASCAL tutoring system. In terms of tutoring system design, it addresses issues associated with user-modelling and selection and development of tutorial strategies.

Department of Computer Science
University of Warwick
Coventry CV4 7AL
United Kingdom

December 1987

LEARNING TO PROGRAM: A COGNITIVE MODEL FOR AN ITS

Iain D. Craig and Robert F. Thomas

Department of Computer Science
University of Warwick
Coventry CV4 7AL
UK

ABSTRACT

Programming is a complex skill acquired only after considerable practise. This paper outlines a hierarchical cognitive model of problem solving and programming. The model accounts for the way students acquire programming knowledge and forms a basis for an integrated problem solving and PASCAL tutoring system. In terms of tutoring system design, it addresses issues associated with user-modelling and selection and development of tutorial strategies.

1. INTRODUCTION

Programming is a complex skill acquired only after considerable practise. This paper outlines a hierarchical cognitive model of problem solving and programming (in this context, for PASCAL programming) which is based on observations of novice programmers and protocol results from experiments. The model accounts for the way students acquire domain knowledge and programming skills, as well as the developmental process that occurs as a novice becomes an expert. It predicts the misconceptions and problems that novices exhibit.

Section 3 of the paper considers the design issues that are addressed by using the model as a basis for an integrated problem solving and PASCAL programming course. The important issues discussed are the optimisation of the knowledge acquisition process and the transition from novice to expert. In terms of tutoring system design, these include user modelling, selection of appropriate tutorial strategies and the graphical representation of problem decomposition to improve understanding.

2 THE MODEL

2.1 Introduction

In this section, the psychological model we are developing is outlined. The model is based on existing psychological theories of problem-solving (Johnson-Laird, 83; Gentner & Stevens, 83; Simon, 79; Gick & Holyoak, 80; Carbonell, 83; Wickelgren, 74) and learning (Norman, 82). It provides a model of the learning process and is intended to account for the transition from novice to expert performance in programming and provides a theoretical basis for the system we intend to develop. It gives an underpinning for the user modelling techniques we intend our ITS to use. We also hope to be able to use the model to optimise the transition from novice to expert performance. Finally, the model suggests a structured presentation of materials: this contrasts with the current practise of lessons that present concepts in a vague fashion and which require concretisation of concepts through programming lessons.

The model we are developing relates to the methodology we adopt in that the point of using the ITS is to develop programming concepts rather than domain (i.e., programming language-specific knowledge - for example, syntax). A more detailed account of the model and its justification is to be found in (Craig & Thomas, 87).

2.2 Model

The model of programming we are developing is based on three collections of mental models which we call the *problem*, *problem-solving* and *domain* models. The models form a hierarchy and each contains a number of smaller mental models to represent the concepts and operators the student possesses. The model hierarchy is shown in figure 1, together with a simple representation of the interactions between models.

The *problem* model is the highest model in the hierarchy. It represents the student's understanding of the problem that has been set. This model contains concepts and operators needed to understand the problem statement in enough detail to begin the problem-solving part of the programming task. In addition to those concepts and operators relevant to the current task, the problem model also has available to it all knowledge acquired for previous problem-solving episodes. The problem model contains real world and other kinds of knowledge brought to bear by students when presented with a new problem statement to turn into a program.

When a problem statement has been understood, it can be converted into a program. The

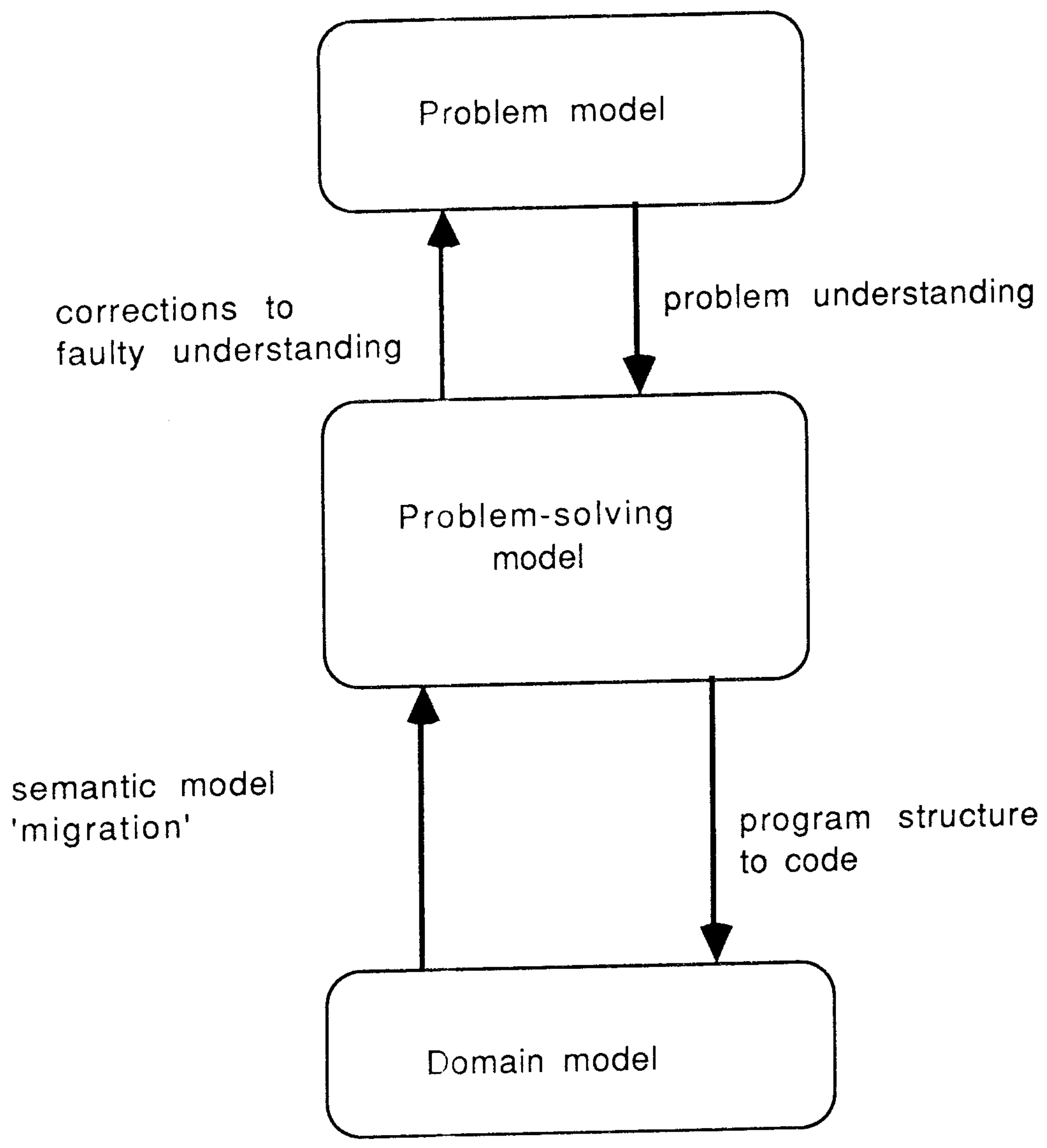
Problem model
corrections to faulty understanding
problem understanding
Problem-solving model
semantic model 'migration'
program structure to code
Domain model

Fig. 1
The model hierarchy and sample interactions

problem-solving model performs the conversion. This model contains a representation of the program design expressed as a subgoal hierarchy. Each major step in structuring the resultant program is represented as a subgoal. Subgoals are created and manipulated by problem-solving operators, each of which is represented by a separate mental model and each of which represents an abstract construction or a general-purpose operator. The representation finally produced by the problem-solving model is a structure which can be directly coded using knowledge contained in the domain model.

The *domain* model contains programming language (in our case, PASCAL) constructs represented in terms of their syntax and semantics. This model is the lowest in the hierarchy depicted in figure 1. The knowledge in this model is again represented as a collection of smaller mental models, one per construct. The domain model represents the basic constructs of the programming language being learned. This knowledge should be used to directly code the structures produced by the problem-solving model. When the subgoal hierarchy has been constructed by the student, knowledge in the domain model is applied to produce a program.

The model just outined is derived from observations of novice programmers and from protocol experiments. It is also based on observations of expert programmers.

It could be argued that only two models (problem and domain) are required to account for novice and expert behaviour. From our observations and experiments, we have found that a range of behaviours is exhibited which indicate that novices have difficulty in mapping problem statements into correct code. We also found that some novices attempt to produce code too rapidly and do not develop a full understanding of the problem before attempting to derive a program structure: as a result, their programs were either incorrect or incomplete - this is a problem that PROUST (Johnson, 86), for example, cannot address.

We have observed, that for novices, problem-solving is shallow and frequently syntax-driven: they do not attempt to use the semantics of programming constructs in their attempts to derive programs from problem statements, but instead attempt to match programming language constructs syntactically against parts of the problem statement. Experts, on the other hand, move from problem statement to code via an intermediate stage in which problem-solving is done in order to produce a (perhaps very detailed) program structure. This gives them the opportunity to set subgoals at a given level of abstraction and to perform a selective breadth-first expansion of subgoals. Typically, expert programmers will select more important or difficult subgoals for attention before they attend to easier or less important ones. Novices tend to engage directly in a depth-first expansion of the subgoals they have generated. Novices exhibit shallow problem-solving because they do not appear to have the abstract concepts and operators available to experts: they work at a level far closer to the code they are to produce. In other words, the problem-solving model possessed by novices is deficient in higher-level constructs than is that possessed by experts.

The development or acquisition of constructs and concepts (mental models, in our terms) is an explanation of the passage from novice to expert. The domain model of a novice contains the syntax and semantics of elementary programming constructs. Initially, we would expert a complete novice to possess only those syntactic constructs encountered so far in the programming course together with a superficial account of their semantics. Since the problem-solving model is deficient, the transition from problem statement to code can only be mediated by comparatively superficial problem-solving. With experience, we suggest, the semantics of constructs becomes better understood (and hence, the mental model which represents each construct's semantics becomes more developed). The development of the semantics allows the construct to be abstracted and used as an operator in the problem-solving model. In addition, more complex constructions (for example, formatted input) are encountered and are turned into abstract operators that can be used for problem-solving.

One significant difference between experts and novices is that experts are able to view programs at abstract levels of representation. For example, an expert might view a module as requiring a set rather than viewing it as some programming language-specific data structure (e.g., an array). In terms of our model, the expert has a mental model of sets in his problem-solving model and is able to use this more abstract operator in program design. The acquisition of such higher-level and more powerful operators and representations in the problem-solving model is part of the way in which that model develops and affords the opportunity to solve problems using deeper inference than does the novice. The acquisition of more abstract operators by experts appears to be based on encountering and using constructs in a variety of different contexts and in different problems: one of our experimental group stated that she found constructs easier to understand after she had used them. The 'migration' of semantics from domain to problem-solving models (where it is used as part of the problem-solving representation) is another facet of this development process.

Because an expert's problem-solving model is more complete and contains more abstract and powerful representations, the interactions between problem and problem-solving models in expert performance are more complex. Novices, as has been noted, tend to produce code before the problem is understood and before a design has been achieved. Experts, on the other hand, spend more time understanding the problem and attempting to produce a design in terms of a subgoal hierarchy. Problem understanding is done using the problem model. Novices, we have observed, engage in a little code-writing and find that they can make no further progress, whereas experts defer coding until as late as possible. We suggest that this is due, in part, to the interactions between the problem and problem-solving models: deficiencies in understanding the problem statement can be detected during problem-solving and before coding if problem-solving is the main activity after coming to grips with the problem statement. In other words, experts tend to use the problem-solving model as a way of isolating deficiencies in their understanding of the problem: when deficiencies are detected, the problem model is used to improve the problem representation so that it can be used by the problem-solving model. Because novices do not possess powerful enough abstractions, they find this process harder to perform.

2.3 Summary of Experimental Results

As has been stated above, thinking-aloud protocols have been taken from members of a small volunteer group drawn from our First Year students. The protocols were taken after we had completed a period of observation during which we watched participants in an elementary course for students who come to Warwick University to read Computer Science but who have no previous experience of programming. We found confirming evidence for our model in the protocols. In particular, we found the effects of problem understanding and problem-solving to be in agreement with our predictions. We also found that subjects whose programming experience was solely that gained as an undergraduate tended to behave in a syntax-directed fashion when confronted with a new problem to solve, whereas those with more experience tended to employ more abstract conceptions of programming concepts.

3. IMPLEMENTATION OF THE MODEL

3.1 Introduction

This section describes a framework for a tutoring system for problem solving and PASCAL programming. It is based on the model outlined in section 1, and addresses important design issues raised by other ITS projects.

3.2 Design philosophy

The design philosophy of our system is built around the whole programming course, rather than the tutoring system, and so contains several different teaching modes.The overall aim is to teach both problem solving skills and program design, which are generic to programming, and PASCAL programming.

The system aims to provide two modes of interaction with the student, a teaching mode and a coaching mode.

The teaching mode teaches PASCAL concepts, adopting a one per lesson strategy (Burton 82). This constrains the teaching domain, so that the system can model the student more accurately in the constrained domain, and can provide more flexibility in its teaching strategies. At the end of the lesson, once the student has learnt and understood the concept, they interact with the computer to create a "concept box", which contains parameters which can be filled to describe the application of the PASCAL concept. When the student has built up a portfolio of boxes and completed the novice PASCAL concepts section, they enter the problem solving mode.This mode provides a learning environment in which the student is given a problem to solve, and is able to develop a plan, and then code it.

The problem solving mode provides a graphics interface which allows the student to build up a goal decomposition diagram, in a hierarchical tree representation. It contains a problem solving coach which provides help, both in structuring the problem, and in teaching problem solving strategies. This approach ensures that the system has an understanding of the student's planning strategy . Since the boxes have been developed during the teaching mode, as a byproduct of the lesson, and since the programming tutor has ensured that the student has developed a good understanding of the concepts, then the problem solving coach has an exact understanding of the semantics of the subgoal boxes. This method has several advantages over other techniques that have been developed to understand or teach problem solving skills. The main advantages are in the following areas.

3.3 User Model

The user model is the the most important, and the most challenging aspect of a tutoring system. It is the system's representation of the student, and hence the basis on which any teaching strategy is selected.

Problems highlighted through research in this area primarily concern the need to be able to pinpoint the level of a student's misconception. Our system is based on a three level model of the process by which a student creates a program, given a specific problem.

During the problem solving mode, when the student is given a question, the notation of this question is an important factor in governing how the student develops a representation, or mental model, of the program (Craig & Thomas, 87). The presentation of different notations for the same question, ie a mathematical question versus a real world example, provides an indication of a student's ability to represent problems. It also provides a method of coaching problem representation, which is a common problem with novice programmers.

During the goal decomposition process, the student uses the boxes they have developed to create a sub goal diagram, or solution plan. Since the system has an understanding of the semantics of the "goal boxes", it can understand the goal decomposition process. Thus it can coach the student with any problems, and also tolerate the many possible ways of planning a solution.

Two ITS systems already developed outline some of the difficulties of understanding goal decomposition.

SPADE (Miller 78) provides a sophisticated graphics interface in which a plan for a turtle graphics program can be developed, using graphics boxes. The student uses a list of defined operators to create box types, but, within the boxes, can type comments in English. As Miller points out, because SPADE cannot interpret the comments, it is severely restricted in its ability to provide advice on goal decomposition and debugging.

Another feature is that the coaching module will contain all possible ways of planning the program. PROUST (Johnson. 86) is a commercially available system that finds non syntactic bugs in PASCAL programs. The system uses a problem description (a subgoal decomposition) which the tutor types into the program in an authoring language. However with all but the most trivial programs, there are several ways of decomposing the problem. Another point which is outlined in section 2 is that novice programmers often misinterpret the question, or start programming without understanding the question. Our tutoring framework would ensure that the student had a proper problem representation, before coding was undertaken.

After the sub goal diagram has been developed, the student can fill in the parameter fields in the individual boxes. The system can then check the student's understanding of the PASCAL syntax and the flow of data.

3.4 Tutoring Module

The task of a tutoring module is to choose the best way of teaching a concept, or correcting a misconception. On a more general level, it should optimise both the acquisition of domain knowledge and the transition from novice to expert.

The implementation of our learning model addresses the student learning issue on a more general level. Teaching PASCAL concepts as mental models, or concept boxes, affords several advantages. It optimises the learning of PASCAL concepts by providing modular concept lessons which allow the student to create and expand PASCAL concept boxes with limited parameters. These boxes can then be used as operators in the problem solving process. In terms of optimising the acquisition of domain knowledge, our models approach allows the student to build upon concepts already learnt, by adding more parameters to the concept box.

As a novice becomes more experienced at programming, he combines simple PASCAL concepts into more abstract operators, performing more complex tasks. This approach aids the student to combine the boxes they have already created into more complex task boxes, allowing them to focus on the problem decomposition process, rather than on the PASCAL syntax.

A problem with teaching strategies has been how to structure the domain to minimise the search space for appropriate tutorial action (Goldstein, 82).

Modularising the lessons and splitting the teaching of programming skills and problem solving skills allows the development of precise tutoring strategies, based on the performance of the student within the limited domain in which he is working. Building on skills and knowledge already learnt will reduce the likelihood of the student displaying complex misconceptions due to the interaction of misunderstandings at several levels. It will also allow the tutor to focus on particular remedial action should the need arise.

3.5 User Interface

In the domain of program design and problem solving, the use of graphics, to show the modularity and hierarchical structure, both of a problem solution and a program structure, is a powerful teaching aid.

During the problem solving mode, an interactive graphics facility will allow the student to select problem solving boxes. and to insert these into the goal hierarchy. This representation allows the student to visualise important problem solving heuristics, as well as understanding the nature of problem decomposition.

4 CONCLUSIONS

This paper outlines a pilot study to develop the basis for a PASCAL Tutoring System. The rationale behind the system is to teach general programming skills, including problem solving and program design, and also to teach the syntax and semantics of PASCAL. The study builds on issues that state of the art ITS systems have outlined and provides a model which can be used to develop lesson material and tutoring strategies and also to provide the basis for a comprehensive student model.

The next stage will be to develop further experiments. Firstly to evaluate the effect of teaching strategies on different levels of misconception and secondly to investigate the use of the question syntax as a method of teaching problem representation skills. Work is also underway on an intelligent graphics interface, allowing a menu based selection of concept boxes.

REFERENCES

Burton R.R. (82) Diagnosing bugs in a simple procedural skill, *in* Sleeman & Brown (82).

Carbonell, J.G. (83) Learning by Analogy: Formulating and Generalizing Plans from Past

Experience, in Michalski, R.S., Carbonell, J.G. and Mitchell, T.M., Machine Learning: An Artificial Intelligence Approach, Tioga Press, Palo Alto, CA, 1983.

Gentner D. and Stevens, A.L. (83) Mental Models, Lawrence Erlbaum Associates, London, 1983.

Gick, M.L. and Holyoak, K.J. (80) Analogical Problem Solving, *Cognitive Psychology*, Vol. 12, pp. 306 - 356, 1980.

Goldstein, I.P. (82) The Genetic Graph: A Representation for the Evolution of Procedural Knowledge, *in* Sleeman and Brown (82).

Johnson W.L. (86) Intention based diagnosis of novice programming errors, Pitman , London.

Johnson-Laird, P.N. (1983) Mental Models, Cambridge University Press, 1983.

Miller M.L. (78) A structured planning and debugging environment for elementary programming, *Int. J. Man-Machine Studies* 11, 79-95.

Norman, D.A. (82) Learning and Memory, W.H. Freeman and Co., San Francisco, 1982.

Simon, H.A. (79) Information-Processing Theory of Human Problem Solving, in Estes, W. (ed.) Handbook of Learning and Cognitive Processes, Vol. 5, Lawrence Erlbaum Associates, London, 1979

Sleeman D., Brown J.S. (eds.) (82) Intelligent tutoring systems, Academic Press, London.

Wickelgren W.A. (74) How to Solve Problems, W.H. Freeman and Co., San Francisco, 1974.