Control Strategies Used By Expert Program Designers.

Steve Lang

Department of Human Sciences, Loughborough University

and

Tom Ormerod

Department of Psychology, Lancaster University

Paper submitted to PPIG 7, Edinburgh January 1995.

Abstract

In this paper we report a study of four expert Prolog programmers designing and coding solutions to an enlarged version of the well-known 'signals' problem. Data are provided showing that experts adopt a predominately structured rather than an opportunistic approach to decomposing design problems. However, the structured approach they appear to adopt is not one of the generally prescribed pure top-down approaches of breadth-first or depth-first problem decomposition. Instead, expert Prolog programmers adopt what we have termed a 'children-first' approach to problem decomposition, in which the relative advantages of breadth-first and depth-first approaches are maximised whilst the disadvantages of these approaches are minimised. We also discuss causes of the few structure-divergent activities that were observed, as well as examining reasons why designers might switch between different structure-congruent strategies.

1. Introduction.

There have been a number of studies investigating the order in which a solution is produced for a design problem. These studies fall broadly into two categories. The first category is where researchers have observed expert designers using a structured approach to the construction of the solution's hierarchy (e.g., Jeffries et al, 1981; Adelson & Soloway, 1985; Ormerod & Ball, 1993). A problem with such studies is that the programmers were generally presented with 'toy' problems such as the signals problem (a program to collect statistics from vehicle survey data) that can be solved with shallow design goal hierarchies. The second group of studies have observed designers solving problems opportunistically (e.g. Ullman et al, 1988; Guindon, 1990; Visser, 1990). These studies have generally focused on larger and more realistic problems in the domain of engineering design (though see Davies, 1992). Researchers have generally measured opportunism as deviations from a prescribed structured control strategy. Where the goal hierarchies were shallow, such deviations cannot be identified easily. On the other hand, it has been argued (Ball & Ormerod, submitted) that weaknesses in the definition of structure-congruent strategies have led some researchers to over-estimate the predominance of opportunism.

Whilst two distinct top-down control strategies, breadth-first and depth-first, have been identified, there is another possible method. We shall call the third approach **children-first.** Superficially, children-first appears to be a mixture of depth-first and breadth-first. The children-first strategy is applied as follows. First the goal is recognised, then the programmer proceeds to describe the goal by recognising all of its immediate sub-goals. So far it appears identical to breadth-first. The final stage is to select one of the sub-goals and apply the above steps until that sub-goal has been completed. When a sub-goal has been completed, the programmer selects the next sub-goal to complete. This continues until the programmer has completed all the sub-goals, and thus has completed the goal itself. This is illustrated in Figure 1 (the designer following an alphabetical order of goal decomposition)

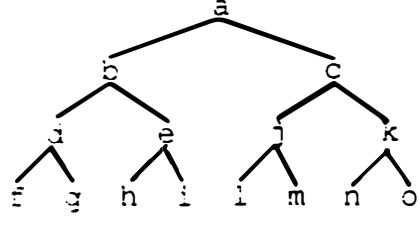

Figure 1: A Children-First design decomposition strategy

Each of these strategies has advantages and disadvantages. A depth-first strategy identifies the fewest potential problems in an emerging design, since each sub-goal in a solution is recognised and completed independent of all other sub-goals. The breadth-first can identify the most potential problems, as each sub-goal is not described until all other sub-goals of the same generation have been recognised. This allows for possible interaction between sub-goals to be discovered. However a breadth-first strategy requires a large amount of cognitive resources to be applied, since the designer must maintain a whole generation of a goal hierarchy at any one time. A designer using a depth-first strategy only focuses on one sub-goal, thereby minimising his/her cognitive load. The children-first strategy takes a middle ground: a sub-goal can be described once its siblings have been recognised. It is not necessary to wait until all the sub-goals of the same generation have been recognised. A designer using a children-first strategy needs only to comprehend all the sub-goals of the same parent.

Two important issues remain to be resolved. On the one hand, it may be that when the goal hierarchy of a programming task is not shallow the designer will work opportunistically. On the other hand, it may be that expert programmers use a structured control strategy that was not recognised by researchers reporting large degrees of opportunism, since it did not conform to one of their prescribed control strategies.

2. The Observational Study.

This study was an investigation of the control strategies used by experienced Prolog program designers. Prolog is a modular language. Therefore the interaction between sub-goals is limited to their siblings. We hypothesise that designers will use a children-first strategy when there is only a negligible possibility that a sub-goal will directly affect another sub-goal which is not its sibling. The source of a problem which a breadth-first strategy can identify but which a children-first strategy can not, is eliminated by using a modular language. In such a situation a solution designed using a children-first strategy is likely to be as good as a solution obtained using a breadth-first strategy, and a children-first strategy would be cognitively easier than a breadth-first strategy. In addition a children-first strategy is likely to obtain a better solution than one arrived at when using a depth-first strategy. Although a depth-first strategy will be cognitively easier to use than a children-first strategy, this is only of secondary importance to an expert designer whose main aim is to design a good solution.

We collected verbal and keystroke protocols from four programmers, all of whom had a number of years of Prolog programming experience in both commercial and academic settings. The task they were given was to produce a solution to an enlarged version of the 'signals' problem (e.g. Ratcliffe & Siddiqi, 1985; Green, Bellamy & Parker, 1987) that was developed such that any solution hierarchy would have at least five levels (see Appendix 1). To determine whether designers were using a children-first strategy or any other strategy we examined the nature of the transitions from one sub-goal to another. Each strategy allows for some transitions and prohibits others. By comparing the actual transitions between goals with those allowed by the various strategies we will be able to determine which strategy best matches the observed transitions. We also examined the nature of verbalisations at transition points, using the coding scheme developed by Ormerod & Ball (1993). This enabled us to identify causes for activity transitions and reasons for switches between solution decomposition strategies.

3. Results.

3.1. Global Control Strategies.

We classified whether a transition from one node to another conformed to each of the three structured control strategies. The table below shows for each designer the percentage of their transitions that conformed to each structure.

	breadth	children	depth	breadth & children	breadth & depth	children & depth	all three
A1	47.43%	50.85%	72.57%	53.14%	80.57%	80.57%	82.86%
A2	56.18%	61.80%	69.67%	62.36%	83.71%	87.08%	87.64%
A3	51.46%	53.40%	69.90%	56.31%	83.50%	81.55%	84.47%
A4	56.67%	59.44%	60.56%	63.33%	79.44%	80.00%	83.89%
$\bar{X}$	52.93%	56.37%	68.17%	58.79%	81.80%	82.30%	84.71%

Table 3: Percentage of moves conforming to structured control strategies.

From the above table it appears that the best fit with the proposed strategies is the use of either breadth and depth-first, or children and depth-first strategies. Presupposing

programmers use all three strategies merely complicates the model without significantly increasing the number of transitions that can be accounted for.

Further analysis shows that the contribution to the breadth-first's score is almost exclusively movement between siblings. This also happens to be part of the children-first's definition. However, the predictions of the two strategies deviate once all the children of a node have been recognised. A designer using a breadth-first strategy would start describing a sibling of the node (if there were any remaining non simple siblings), while a designer using a children-first strategy would start describing one of the children of the node (if they are not simple). Inspection of the keystroke protocols showed that whenever there were non-simple children of a node that had just been described and the designer was not employing a depth-first strategy, the designers always chose to describe one of these non-simple children rather than describing a non-simple sibling of the node. This supports the assertion that designers used children-first and depth-first approaches rather than breadth-first and depth-first approaches.

3.2. Structure-Divergent Activities.

On average 82% of a programmer's activity transitions can be accounted for by him/her using a mixture of children and depth-first strategies. The remaining 18% of transitions are divergent from these two top-down structured control strategies.

These structured-divergent transitions were analysed further by examining the verbal protocols and the goal hierarchies to determine their nature. They were found to consist of 8 types: (1) a transition to a node whose parent has not been coded, but which has been recognised; (2) a transition that jumps back to return to a structured approach after a divergence; (3) debugging, a transition to amend an erroneous node; (4) a transition to capitalise on an analogy; (5) a transition to implement a pre-requisite for the current goal; (6) selecting a goal based on its ease of implementing; (7) bottom-up, a transition to implement a higher level goal; and (8) a transition to implement a post-requisite for the current goal.

Parent not Coded	5.52%
Jump-Back	3.04%
Analogy	1.89%
Debugging	1.83%
Easy Goal	1.51%
Pre-requisite	1.36%
Post-requisite	1.32%
Bottom-up	1.23%
Total	17.70%

Table 4: Distribution of the 8 Types of Structured-Divergent Transitions

3.3. Factors in Switching Between Structured Strategies.

From examination of the final goal hierarchy of each programmer's solution along with analysis of relevant verbal and keystroke protocol sections, we determined two factors that contribute to switches between depth-first and children-first strategies. These were: (1) whether the sub-goals to be developed were disjunctives; and (2) the difficulty of designing the sub-goals. As a measurement of a sub-goal's difficulty, we calculated its complexity which is the number of generations below the sub-goal.

Whenever the designers encountered disjunctives they employed a depth-first strategy. In the situations where the designers were confronted with conjunctive sub-goals their choice of strategy depended on the difficulty of the sub-goals. Designers were likely to use a children-first strategy when the sub-goals were difficult, but a depth-first strategy when the sub-goals were easy. Using the Wilcoxon Rank Sum test at the 99.99% level, we found ($W'_s(15,15) = 143.5$) that the complexity of the sub-goals according to our chosen measure of goal complexity was significantly lower when using a depth-first strategy than when using a children-first strategy.

4. Discussion.

This study has demonstrated that expert Prolog programmers do adopt a predominately structured approach to the design of programs, for both simple and more complex programming tasks. They appear to use a combination of children-first and depth-first strategies, rather than a breadth-first strategy. We also have identified the causes of structure-divergent behaviour. From our classification, we would argue that the majority of structure-divergent activities reflect the repair of a structured approach in response to a local divergence from the goal hierarchy. Typically, these reflect the later coding of design components that had been recognised earlier as part of a structured approach.

Why do Prolog programmers not use a breadth-first strategy? The designer seems to choose the most economical strategy for producing a good design. The economical cost of a strategy involves: the difficulty of using the strategy; and the likelihood of the strategy producing incompatible goals against the perceived cost of producing and rectifying incompatible goals .

> economic cost = difficulty of strategy + (chance of incompatible goals × cost of producing and rectifying incompatible goals)

Exact parameters for this formula have yet to be established, but they are probably dependent upon task, experience, language and design environment. This formula is

in some respects similar to the proposals offered by Visser (1990) to account for opportunistic design activity. However, whilst she proposes cognitive cost as the primary motivation for choosing which design goal to focus upon at any one time, like Ball & Ormerod (submitted) we argue that experts also evaluate the longer term cost-effectiveness of a design strategy in choosing a method of problem decomposition.

Is this result generalisable to experts in other programming languages? The likelihood for a children-first strategy and a breadth-first strategy of producing incompatible goals is the same if the heuristic of modularisation is adopted. A children-first strategy is more likely to produce incompatible goals than a breadth-first strategy if an alternative heuristic is used. Within the modular language Prolog, a children-first strategy and a breadth-first strategy will identify the same sources of problems. Whereas in a non modular language such as C, a breath-first strategy will identify more sources of problems than a children-first strategy.

The cost of producing and rectifying incompatible goals is generally dependent on the difficulty of the goals. Within this study the difficulty of a goal was measured by its complexity, the depth of sub-goals below the goal. If a goal consists of sub-goals which are not complex, the designer may choose a depth-first strategy as the cost of producing and recovering from a mistake is low. If a goal consists of disjunctive sub-goals, the dependency of the sub-goals on each other is low. It is unlikely that a depth-first strategy would produce incompatible disjunctive sub-goals, as their dependency on each other is low.

Although this observational study has highlighted factors which may influence the designer's choice of control strategy it remains to be seen whether this will hold in other circumstances, such as designers using a different programming language. Furthermore it remains to be seen whether forcing a designer to adopt a strategy which s/he would not have chosen voluntary enhances or reduces his/her ability to produce a good artefact.

References - to follow

Appendix - to follow

Acknowledgements - to follow