An Investigation Into Strategies Employed In Solving A Programming Task Using Prolog

J Siddiqi, *Computing Research Centre, Sheffield Hallam University*
B Khazaei, *School of Computing and IT, University of Wolverhampton*
R Osborn, *Computing Research Centre, Sheffield Hallam University*
C Roast, *Computing Research Centre, Sheffield Hallam University*

Abstract

This paper highlights the carry over effects in changing from a procedural to a declarative approach. The results of a case study into programming in Prolog for a relatively simple problem is reported. This paper describes the different methods of solutions that these subjects used to solve the problem and argues that they can be explained on the bases of strategies used for problem decomposition and the choice of data representation. It argues that the methods of solutions used suffer from a "carry over effect" based on a procedural approach. In particular, that the choice of data representation used appears to be more important than the paradigm used.

1. Introduction

Programming in a logic based paradigm makes use of predicate logic which allows one to state a programming solution in a declarative form, and it is argued that this is more natural than a procedural form for a large number of problems [1]. Some cognitive scientists [2] have questioned the issue of naturalness of declarative forms.

>From a human factors point of view the problem of "PD-programmers" (ie those traditionally trained and experienced in a procedural approach) learning Prolog programming is twofold. One the one hand, they are required to express their solutions in a logic paradigm which is a novel idea because they are used to "procedural thinking". On the other hand, they would need to know and consider the "control flow" of a logic based language which may or may not be identical to procedural features they are familiar with. This combination in some cases can be confusing. At present there is some empirical evidence reporting this phenomenon.

One study reports that programmers who have been trained in and used programming principles based on the procedural style have difficulties in adapting to the declarative style [3]. We believe this is because these programmers seem to continue to use the principles of the former rather than the latter style. It is not unreasonable to expect this because it is known that people have strong tendency to apply known methods rather than learn new methods. Therefore, we argue that for Prolog programming the underlying execution mechanism used by PD-programmer relies heavily on procedural/operational "thinking". This tendency produces what we call "carry over effects" which in certain circumstances can lead to misconceptions. There is an absence of detailed published empirical evidence which elaborates on these carry over effects. The aim of our invention is to provide an insight into the crucial issues that need attention in order to ease the transition of PD-programmers from a procedural style to a declarative style of programming. In so doing we will highlight the dual procedural and declarative models used by PD-programmers.

Section 2 details the specifics of a case study and the results of this are presented in Section 3.

2. Case Study

32 second year under-graduate computer science students undertook an assessment for a one semester module on functional and logic programming.
The students for nearly two academic years, had received training in and used a procedural approach to programming. The exercise was to produce a Prolog program for the "Bridge Hand Problem". The statement of the problem is as follows;

Write a Prolog program which accepts as input a representation of a bridge hand consisting of 13 cards supplied in random order. The program is required to produce as output:

(a) the hand of cards arranged in descending order by rank within each
(b) the points value of the hand (counting 4, 3, 2, 1, for Ace, King, Queen and Jack resp.)

An example output is as follows:

CLUBS K 10 9
DIAMONDS J 9 4 3
HEARTS A Q 10 8 2
SPADES 7

POINTS VALUE = 10

The Bridge Hand problem was the subject of a previous observational study into designer behaviour involving programmers using a procedural approach [4]. The choice of problem was therefore well suited for an initial comparative study between procedural and declarative paradigms.

Although the majority of the students had difficulties in providing a complete working solution to this problem, sixteen of them succeeded in producing comprehensive working programs. The analysis carried out were similar to that of Siddiqi [5] that is the solutions were compared to identify distinct approaches. The classification chosen was in terms of decision made concerning "the choice of representation". This led to subjects attempts being classified into two solution types. One in which the subjects chose to transform the input representation to the desired output representation (ie an ordered set of values) by means of an explicit sort routine, hereafter referred to as transform type. The method of solutions involves splitting the hand into four newly created lists according to suits. Each card in the hand is inserted into the appropriate list according to its value.

The other in which subjects chose to process the input representation in its original form with the honour cards being revalued so as to facilitate the use of the in-built sort routine. This solution, hereafter referred to as patch it type, involves using a "patching" routine to convert the sorted list into the desired output. In terms of Siddiqi's previous work [5] transform type represents a "data driven" approach, because the primary focus is on processing the data stream. Whilst the patch it type represents a "goal driven" approach, because the goal is to "sort" the hand using the built in sort routine.

From the 32 attempted solutions 24 (75%) were of the patch it type. The most likely explanation for this is that subjects were attempting to use a "do what you can and make the rest fit around it". A strategy reported by Siddiqi in the study of subjects using a procedural approach [5]. For the Prolog solution, subjects recognised the benefits of making use of the in-built sort routine (ie an island of certainty) and adding "patches" to facilitate this (fitting the rest around the island). It is hypothesised that the students who provided the transform type solution had used a data driven approach and did not rely on the built-in sort routine.

3. Discussion

There are two important observations, based on the case study, that can be made. First, both the decomposition strategies employed namely data-driven and goal driven are direct carry over effects from procedural programming, and there appears to be little evidence supporting the use of "predicate logic" and/or declarative style in these approaches. It would also appear that the application of these strategies is not carried out in a top-down manner. Furthermore, as was the case for our study of procedural programming [4], the application of these strategies can be more readily explained in terms of using "island driving" that is forming an "island of certainty" around what you can do and then extending it in multi-directional manner by taking the rest around this island.

Second, which concurs with the results we obtained in our protocol analysis study [4] where subjects adopted a procedural approach, the choice of data representation is a determinant factor in shaping algorithm design. Furthermore, working in declarative style does not appear to significantly reduce the strong tendency towards simplistic representations because as mentioned previously 76% "chose" the simpler but at the same time inappropriate representation for a hand. Further evidence of this propensity is choice of representation of a card, a significant proportion again chose the most obvious representation which is less appropriate for the needs of the processing requirements namely a nested list rather than a linear list.

In conclusion, working in the declarative paradigm does not prevent the strongly observed tendency, when working in a procedural paradigm [4,6] towards simplistic representations and inclinations towards performing problem decomposition on the basis of "do what you can and make the rest fit around it". Therefore, it would appear that choice of data representation and decomposition strategies appears to be more important than the programming paradigm used. Moreover, we assert that subjects behaviour can be more readily explained in terms of carry over effects due to procedural approach rather than a declarative approach.

A further study was conducted which identified common misconceptions believed to be direct results of "carry over effects" imperative programming. These are currently being analysed, some preliminary results relating to the most frequently observed occurrences of these misconceptions will be presented.

References

[1] Kowalski, R., (1979), "Logic for Problem Solving", North Holland Inc.

[2] Taylor, Josie, (1984), "Why Novices Will Find Learning Prolog Hard?" CSRP.044, University of Sussex.

[3] Someren Van. M., (1984), "Misconceptions of Beginning Novice Programmer, TheAcquisition of Expertise", Department of Psychology,University of Amsterdam. Memo 30.

[4] Siddiqi, J.I.A., Khazaei, B. "Models of Programmer Behaviour: A Comparative Study." The Twelfth Annual International Computer Software and Applications Conference.I.E.E.E. COMPSAC 88, 141-146.

[5] Siddiqi, J.I.A., (1984), "An Empirical Investigation Into Problem Decomposition Strategies used in program design", Ph.D Thesis, University of Aston in Birmingham.

[6] Ratcliff, B., Siddiqi, J.I.A., (1985), "Problem Decomposition Strategies Used in Program Design", International Journal of Man-Machine Studies 22, 77-90.

All correspondence should be directed to:

Jawed Siddiqi
Computing Research Centre
Sheffield Hallam University
Hallamshire Business Park
100 Napier Street
Sheffield
S11 8HD

Tel: 0114 2 533781
Fax: 0114 2 533161
Email: J.I.Siddiqi@SHU.AC.UK