WHAT DO NOVICE PROGRAMMERS KNOW ABOUT RECURSION

Hank Kahney
Human Cognition Research Laboratory
The Open University
Milton Keynes, England

ABSTRACT

Recent research into differences between novice and expert computer programmers has provided evidence that experts know more than novices, and what they know is better organized. The conclusion is only as interesting as it is intuitive. This paper reports an experiment which was designed to determine precisely what novice programmers understand about the behaviour of recursive procedures, and exactly how their understanding differs from an expert's understanding of the process. The results show that different novices understand, or misunderstand, different things. Implications of the findings are discussed with respect to other research into novice and expert programming performance.

1.0 INTRODUCTION

A course in Cognitive Psychology offered to students at the Open University is highly favourable to computer models of cognitive processes, and in order to familiarize students with computer terminology a friendly software environment called SOLO has been designed (Eisenstadt, 1978). Solo is a LOGO-like database manipulation language with a handful of primitives for searching and side-effecting an assertional database of associative triples. Students are very quickly introduced to concepts such as recursion. When taught about recursion an example program is provided and the student is led through several pages of detailed description about the way the program behaves when operating on a particular database. However, analyses of more than a hundred recursive procedures written by our students indicate that many have some difficulty or even find it impossible to design correct programs.

The task discussed in this paper was designed 1) to test the hypothesis that novices and experts differ in terms of their respective models of recursion as a process, even after a detailed introduction to the behaviour of a recursive procedure, and, 2) to try to discriminate the models of recursion which novices actually do come to possess.

The conceptual model presented to students in the SOLO Programming Manual defines recursion as a process that is capable of triggering new instantiations of itself, with control passing forward to successive instantiations and back from terminated ones. This is the model of the recursive process that experts are hypothesized to have.

Students, on the other hand, are hypothesized to have a 'looping' model of recursion. That is, they view a recursive procedure as a single object instead of a series of new instantiations, having the following features: 1) an 'entry point', the constituents of which are the procedure's name and a parameter slot; 2) an 'action part', which is designed to add information to the database; 3) a 'propogation-mechanism' for generating successive database nodes and feeding the values of these successive nodes back to the 'front part', or 'entry point' of the procedure. In this Looping model, the parameter '/X/' is treated like a box which holds a value which is displaced by subsequent values.

2.0. THE BEHAVIOUR OF PROGRAMS PREDICTED BY THE DIFFERENT MODELS

The hypothesis about differences between novice and expert models of recursion was tested by presenting Subjects with the Questionnaire of Figure 1, below. As may be seen, the Questionnaire contains two programs, called SOLUTION-1, and SOLUTION-2,

respectively. These programs are critical to determining a Subject's model of recursion. The text of the Questionnaire is given in Figure 1. (The text has been minimally edited for reasons of lack of space. The difference is that a third Solution - a program that would not achieve the required result - has been deleted.)

---------- PROPAGATING INFERENCES ----------

Recently I needed a programme which would make the following inference: if somebody 'X' has 'flu' then whoever 'X' kisses also has 'flu,' and whoever is infected spreads the infection to the person he or she kisses, and so on. Starting with the database given in Figure A, I needed a programme which would change the Figure A database into the Figure B database.

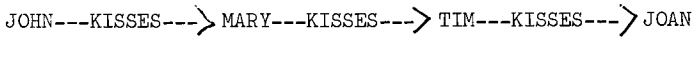

FIGURE A.

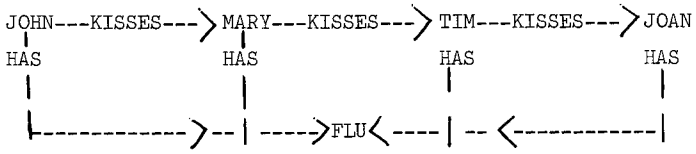

FIGURE B.

I have been provided with two solutions to the problem, both called 'TO INFECT /X/' and these are labelled SOLUTION-1, SOLUTION-2, below. I want you to consider each programme in turn and say (A) whether or not the programme will do what I want it to do, and (B) if it will, say how it does it (in your own words), or, if it won't, say why it doesn't (again in your own words).

SOLUTION-1:

```
TO INFECT /X/
1 NOTE /X/ HAS FLU
2 CHECK /X/ KISSES ?
2A If Present: INFECT * ; EXIT
2B If Absent:  EXIT
DONE
```

SOLUTION-2:

```
TO INFECT /X/
1  CHECK /X/ KISSES ?
1A If Present: INFECT * ; CONTINUE
1B If Absent:  CONTINUE
2  NOTE /X/ HAS FLU
DONE
```

Please write your answers on the pages provided overleaf.
Thank you for cooperating.

---------- FIGURE 1. ----------

SOLUTION-1 and SOLUTION-2 would both achieve the required output database. SOLUTION-1 works by side-effecting the database on the node first given as the argument to INFECT (= JOHN), and then generating the next node on the 'KISSES' list, which triggers the recursion. SOLUTION-2 works by creating a stack of bindings for /X/, ie, (JOHN MARY TIM JOAN) and side-effecting each on return from the recursive creation of the list (ie., side-effects the listed nodes in reverse order).

2.1 The Effects Of Running The Programms Under The Copies and Loop Models.

Call the conceptual model presented in the SOLO Programming Manual the 'Copies' model of recursion, and the model hypothesized for students the 'Loop' model of recursion. Programmers possessing the Copies model would reason that the programs behaved as described in the previous paragraph. Under the Loop model, however, a programmer would argue that the first Solution would be okay, but not the second. In SOLUTION-1, the parameter would be bound to JOHN, and at the first step of the program the description 'HAS FLU' would be added to that node in the database. 'MARY' would be generated by the pattern-match at Step 2, triggering the 'Loop'. The Infect procedure would then start over again with 'MARY' displacing 'JOHN' in the parameter slot. And so on. In SOLUTION-2, the parameter would be bound to 'JOHN'. The pattern-match at Step 1 would generate 'MARY', triggering the Loop. The Infect procedure would then begin again with 'MARY' displacing 'JOHN'. On the next pass through the Loop 'TIM' would displace 'MARY'. Finally, 'JOAN' would displace 'TIM'. Also, because 'JOAN' is not linked to any other node through the 'KISSES' link, the pattern match would fail, and the program would proceed

to Step 2, with the result that the description 'HAS FLU' would be added to that node in the database.

Strong evidence for possession of the Copies model, then, would be selection of both SOLUTION-1 and SOLUTION-2 as correctly designed programs for the task in hand, plus some comment on the order in which the side-effect to the database occurs when SOLUTION-2 is run; since the side-effect occurs as the recursion unwinds, one would expect anyone who recognized this fact would also mention it.

Strong evidence for possession of the Loop model would be; 1) selection of SOLUTION-1 as correctly designed, and, 2) rejection of SOLUTION-2 on the grounds that only JOAN would be affected by running this program.

In all cases, the Subject's reasons for selecting or rejecting one and another of the programs should be examined for direct evidence about the model possessed.

METHOD: The full intake of students (approximately 90) for the first week of the Cognitive Psychology Summer School (July, 1981) were given the Questionnaire (Figure 1, above) and asked to fill it in at their leisure and to return it to the experimenter. Students' previous experience of programming is not known. Nine experts also acted as Subjects in the experiment. The experts were Tutors in Artificial Intelligence at the D303 Summer School, with the exception of one Subject who was a research assistant in the psychology laboratory, and who had had a year of experience in writing Lisp, PASCAL and SOLO programs. In the remainder of this paper, novice Subjects who took part in the experiment will be referred to as 'Respondents'.

RESULTS: Eight of the nine experts selected both SOLUTION-1 and SOLUTION-2, and one selected only SOLUTION-1 as the programs that would achieve the required output for the Questionnaire task. In all cases, the comments of the Experts selecting SOLUTION-2 indicated that they had a Copies model of recursion.

Of the 90 or so novices who were given the Questionnaire, 30 completed and returned it. The number of Respondents who chose different programs (or combinations of programs) as behaving as required is given in square brackets in Table 1, below.

1) None of the Programs would behave as intended [4].
4) Only SOLUTION-1 would work [16].
5) Only SOLUTION-2 would work [5].
6) Both SOLUTION-1 and SOLUTION 2 would achieve the intended output database [3].

TABLE 1.

Table 2 shows the numbers of novices (N) and experts (E) who chose either SOLUTION-1, or both SOLUTION-1 and SOLUTION-2 as programs that would achieve the required effect. (The results include data on selection of the third program which was deleted from Figure 1, above). All other responses were collapsed into the category named OTHER. The difference in selection between novices and experts is highly significant (chi-squared = 21.40 p .001).

Table 3 shows the number of novices and experts who selected only either SOLUTION-1, or SOLUTION-2 as correct solutions to the problem (the category OTHER has been removed). Novices chose SOLUTION-1 and SOLUTION-2 significantly less often than experts (chi-squared = 10.78, p .01).

	N	E
SOLUTION-1	16	1
SOLUTION-1 & 3	3	8
OTHER	11	0

chi-squared with 2 degrees of freedom =21.40
contingency coeff, = .59

TABLE 2

	N	E
SOLUTION-1	16	1
SOLUTION-1 & 3	3	8

chi-squared with 1 degree of freedom = 10.78
cont.coeff, = .52

TABLE 3

DISCUSSION: These results suggest that just over half the Respondents have adopted the Loop model of recursion (selected SOLUTION-1 and rejected SOLUTION-2), and that only three of the 30 Respondents have acquired the Copies model (selected both SOLUTION-1 and SOLUTION-2). Six of the Respondents appear to have understood little, if anything, about any of the programs (Respondents in categories 1,2, and 3 in Table 4.1). In order to determine more precisely how the different Respondents thought the programs behaved, an examination of their reasons for selecting and rejecting programs were examined. Limitations of space preclude examination of the protocols in this paper, but any interested reader can find a detailed discussion of this data in Kahney (1982). However, the findings from the analysis of the protocols will be briefly summarized in the next few sections of the paper.

2.2. The Strong Evidence For The Copies Model.

Three Respondents out of thirty selected both SOLUTION-1 and SOLUTION-2 as correct programs, which was argued to be strong evidence for possession of a Copies model of recursion. However, the comments made by two of the three Respondents cast doubts on their level of understanding of either of these solutions. In the end the data suggest that only one in thirty Respondents after their initial training in SOLO programming, has acquired an expert's understanding of recursion, the Copies model.

2.3. The Strong Evidence For The Loop Model.

Of the sixteen Respondents who selected SOLUTION-1, four rejected SOLUTION-2 on the grounds that only JOAN would get 'flu.' Since it has been argued that this is the strong evidence for possession of the Loop model, the figures indicate that just 13% of all Respondents have this model of recursion. Three other Respondents selected SOLUTION-1, and rejected SOLUTION-2 on the grounds that 'only' one person would get 'flu', but the 'person' was 'JOHN', not 'JOAN'. These Respondents seem to have a 'matching bias'. They may believe that the original value of the parameter ' /X/' is saved rather than displaced by subsequent values generated by the pattern match.

2.4. The Weak Evidence For The Loop Model.

Most of the Respondents chose SOLUTION-1 as the only program that behaved as required, which is considered to be only weak evidence that the Loop model has been acquired by just over half (53%) of those who filled in the Questionnaire. Unfortunately, most of the comments made by these Respondents are less informative than necessary to work out their model in detail. Only one Respondent made specific reference to looping as a mechanism of recursion. (This alone, of course, does not by itself mean that this Respondent actually has the Loop model). While it is reasonable to assume that some of these Respondents have the Loop model (if their comments had been elaborated) there is good evidence that many have rather idiosyncratic models of the behaviour of recursive procedures.

3.0. OTHER MODELS OF RECURSION.

A possibility not yet considered, of course, is that students (or some of them) have no model of recursion, or a model different from either the Copies or the Loop model. Some of the Respondents clearly do not understand recursion at all.

3.1. The 'Null' Model.

In this class would fall thos Respondents who said that none of the programs would work. Typical of the comments made by these Respondents was this:

> "Won't work. This is because as a definition of the procedure 'infect', it can hardly use that very procedure as part of the initial definition. This would probably be refused by the computer."

3.2. The 'Odd' Model.

Several Respondents had acquired the notion that the flow of control statement, rather than the results of pattern matching, acts as the stopping rule for recursion. (Such notions may be quite common. In experiments in which we have videotaped programming sessions by six novice programmers we have one who thought that the flow of control statement 'EXIT' caused a recursive procedure to terminate too soon, one who thought it would never stop if the flow of control statement 'CONTINUE' was present, and one who believed that recursion could only work if there was some sort of 'active' relationship between database nodes used in pattern matching).

3.3. The 'Syntactic', 'Magic', Model.

Hints that some students base their judgements about the behaviour of programs in the syntactic structure of the program come from many of the comments made by different Respondents. These Respondents appear to be sensitive to the position of the different program segments, and make their predictions about the behaviour of programs on this basis rather than on a model of the way the program actually behaves.

SUMMARY: The Copies model is not a viable candidate for what our students know about recursion. Only three out of thirty Respondents on the Questionnaire showed evidence for the Copies model, and two thirds of this evidence did not stand up to scrutiny. On the other hand, only four of the Respondents appear definitely to have the Loop model, on the evidence of the strong indicants of that model. Thus, what the Respondents know about recursion can be accounted in terms of either the Copies or the Loop model in only five out of thirty cases. The comments of many of the Respondents who have been classified as providing only weak evidence for the Loop model suggest that they have acquired something other than either the Loop or the Copies models.

4.0. CONCLUSION.

A range of abilities is demonstrated in these results. Novices can be distributed into different classes according to the internal structure of the individual concepts they have acquired. Previous studies have attempted to show that novices and experts differ in the amount and organization of knowledge possessed by the different groups. Such effects are usually demonstrated in tasks which bear no relationship to the usual activities of

programmers (e.g., Adelson, 1981). McKeithen et al. (1981) have provided evidence that novices and experts differ in the extent to which they conceive of particular computing constructs as being related. A major problem with the analysis is that once the beginners and novices learn which concepts 'go together' their performance on tasks such as that devised by McKeithen et al, will be indistinguishable from the performance of experts. But this will not mean that they know what experts know. A task such as that reported in this paper is designed to get at the knowledge novices and experts have about individual computing constructs, their process knowledge.

The data show that at least some novices - probably a quarter - can, after a fairly brief training period in SOLO programming, identify and mentally simulate the behaviour of recursive procedures. If a person has a mental model of a process, even if it is at variance with the conceptual model of the process, he will be able to make predictions about the behaviour of the process, although perhaps not all the behaviour, and perhaps inaccurately (Norman, 1982; Collins & Gentner, 1982). That is, students who are able to develop a Loop model of recursion will be able to design procedures in terms of the model and understand unfamiliar programs by mentally simulating their behaviour in terms of the model. More importantly, possession of a model provides a person with a basis for debugging the model when confronted with a counterexample (Jeffries, 1982).

An important implication of the findings is that different novices, with different models of recursion, may produce exactly the same programmed solution to particular problems. These solutions would be the same solutions devised by experts. On the surface there would be no differences in the programs, suggesting, wrongly, that there are no important differences in the knowledge underlying the performances. We have seen this occur in our own laboratory, with students who have been videotaped solving programming problems for us. The result casts doubt on the assumption that particular knowledge is needed in order to solve certain problems, and that, if a certain problem is solved, the solver has particular knowledge (as in Ehrlich & Soloway, 1982).

REFERENCES:

Adelson, B. Problem solving and the development of abstract categories in programming languages. Memory and Cognition, 1981, 9, 422-433.

Collins, A. & Gentner, D. Constructing runnable mental models. Proceedings of the Fourth Annual Conference of the Cognitive Science Society, Ann Arbor, Michigan, 1982.

Ehrlich, K. & Soloway, E. An empirical investigation of the tacit plan knowledge in programming. New Haven, Conn.: Technical Report 82-236, Department of Computer Science, Yale University, 1982.

Eisenstadt, M. Artificial Intelligence Project. Units 3 and 4 of Cognitive Psychology a third level course. Milton Keynes: Open University Press, 1978.

Jeffries, R. A comparison of the debugging behaviour of expert and novice programmers. Paper presented at the AERA annual meeting, March, 1982.

Kahney, A. An in-depth study of the cognitive behaviour of novice programmers. Technical Report No.5. Human Cognition Research Laboratory, The Open University, Milton Keynes, England, 1982.

McKeithen, K.B. Reitman, J.S. Rueter, H.H. & Hirtle, S.C. Knowledge organization and skill differences in computer programmers. Cognitive Psychology, 1981, 13.

Norman, D.A. Some observations on mental models. In D. Gentner & A Stevens (Eds.), Mental Models. Hillsdale, N.J.: Lawrence Erlbaum Associates, 1982.