Mental Models of Recursion: Investigating Students’ Understanding of Recursion

Tamarisk Scholtz and Ian Sanders
School of Computer Science
University of the Witwatersrand (Wits)
Johannesburg, South Africa
scholtzt@cs.wits.ac.za, ian@cs.wits.ac.za

ABSTRACT

Mental models of recursion provide some idea into a student’s understanding of recursion. However there has been concern regarding whether viable trace mental models of recursion show students’ true understanding of recursion. We have conducted an investigation to further examine the understanding of recursion of students with viable trace mental models. The investigation looked at students’ understanding of the termination of a recursive function, their descriptive mental models and their ability to generate a recursive function. This research provides evidence to show that trace methods are essentially mechanical processes that can allow students with little understanding of recursion to correctly evaluate a recursive function but that students do not fully understand recursion and in particular have difficulties with the passive flow. Based on the results of the study, this paper discusses possible changes that can be made to our teaching in order to more effectively teach recursion to first year students.

Categories and Subject Descriptors
D.3.3 [Programming Languages]: Language Constructs and Features—Recursion; K.3.2 [Computers and Education]: Computer and Information Science Education—Computer Science Education

General Terms
Algorithms

Keywords
Recursion, Mental Models

1. INTRODUCTION

In the School of Computer Science at the University of the Witwatersrand (Wits) we have used students’ traces of the execution of recursive functions in an attempt to understand our students’ understanding of the two flows of control of recursion [4]. These traces are categorised into mental models of recursion which we believe give us an understanding of how the students think about recursion. A mental model is viable if it allows the student to deal with recursion. It was found that students can have viable mental models (based on their traces and henceforth termed trace mental models) which allow them to correctly compute the output of a recursive function. This seems to indicate that these students do indeed understand recursion. We are, however, concerned whether these trace mental models reveal the students’ actual understanding of recursion or whether they show that the students have simply mastered a mechanical process for tracing recursive algorithms. In this paper we discuss an experiment which we conducted to investigate this by examining the understanding of recursion of students with viable trace mental models.

The students in the 2009 first year class at Wits were given a test that examined their ability to trace three recursive functions, prove whether a recursive function terminates and provide written descriptions of the execution of a recursive function. Thereafter 15 students who had viable trace mental models of recursion were selected to be interviewed. In the interviews they were asked to evaluate a recursive algorithm that drew a pattern and asked to design a recursive algorithm that printed a specific pattern. The study provided sufficient evidence to prove that trace methods are mechanical processes and that effective application of the trace methods does not necessarily imply real understanding of recursion. The results of the study have also revealed that most students do not fully understand the passive flow of recursive functions.

A factor that affected these results is that our first year students are shown simple recurrence relation defined or list manipulation recursive algorithms. Originally we introduced recursion to students using the factorial or Fibonacci numbers algorithms but we realised that tracing these algorithms did not require an understanding of the passive flow and resulted in the development of non-viable mental models [11]. We then changed to introducing recurrence relation based algorithms and more complex list manipulation algorithms [11]. This study discovered that these recursive functions are also easily computed by trace methods. Additionally it was found that students do not gain an understanding of the passive flow, since these algorithms do not
explicitly show the point at which the algorithm suspends and it’s return to that point in the passive flow.

This paper also highlights some changes that can be made when teaching recursion to students. Specifically, students should be shown a variety of recursive functions, such as embedded recursive functions with commands that have to be executed after the recursive call as these will allow students to “see” the passive flow of the recursive function, as the function returns to the point where it was suspended to execute the remaining commands.

The next section (Section 2) describes the details of the study that was conducted to further examine students’ understanding of recursion plus the results of the study. Section 3 discusses the implications of the study. In particular that trace methods are mechanical processes which cannot be used as indicators of real understanding (Section 3.1) and that students do not fully understand the passive flow of recursion (Section 3.2). It also offers some suggestions for teaching recursion to first year students (Section 3.3) – this includes a proposal for showing a variety of recursive functions. Lastly a summary of this article is given in Section 4.

2. INVESTIGATING STUDENTS’ UNDERSTANDING OF RECURSION

The copies mental model was found to be a viable mental model of recursion, as students with this model show an understanding of the active flow, limiting case evaluation and passive flow of the recursive function [5, 4]. Kahney [5] cautions that possession of a copies mental model is not sufficient to allow one to determine what a student really knows about recursion since students can make predictions about the behaviour of a recursive function without fully understanding recursion. We were concerned whether the fact that a student showed a viable trace mental model of recursion did, in fact, mean that they had an indepth understanding of recursion. Sections 2.1 and 2.2 below describe a study that was conducted to further examine our students’ knowledge of recursion.

2.1 Data Collection and Analysis

123 first year students at Wits were given a test to determine their understanding of recursion. The test examined the students’ ability to trace recursive functions, prove when a recursive function would terminate and explain the execution of a recursive function. The test consisted of three questions which asked students to trace the output of the recursive functions for given input values. All three main algorithms were list manipulation recursive functions. The first function removed the first element each time and multiplied the list elements together starting from the end of the list. Similarly the second algorithm removed the first element of the list and added them together, with the exception of the last element (the limiting case) which was input into a second function that returned either 1 or the result of summing from 1 to the value of the last element. The third recursive function used a second algorithm to compute the sum of the powers of 2 of the list elements – see Figure 1. Students’ solutions to these questions were categorized into the trace mental models of recursion defined by Götschi et al [4, 11, 10].

Students were also asked to prove that Algorithm 1 given in Figure 1 would terminate for any list of non-negative numbers. A question from the final examination of the students’ Fundamental Algorithmic Concepts (FAC) course was also analyzed. This question asked them to state when a recursive function would terminate. The function searched for an element in a list. It considered both cases, when the element was found in the list and when it was not found in the list. These questions examined whether students understood that a recursive function reaches the limiting case then completes the passive flow before it terminates. Levy and Lapidot [6] found that analyzing students’ discourse of recursive functions can reveal their understanding of the concept. Therefore we have looked at students’ written descriptions of the execution of Algorithm 1. The study aimed at identifying mental models of recursion from students’ proofs of the termination of a recursive function and descriptions of the execution of a recursive function. These descriptive mental models were categorized by determining whether students identified the active flow, limiting case and passive flow of the given recursive functions.

A more indepth study was then conducted. Here 15 students, who showed a viable trace mental model of recursion in each of the three trace questions, were interviewed/observed. These students were observed as they solved two graphically related recursive problems. The first question required students to evaluate the algorithm given in Figure 2, that printed a shape. The second question required students to design a recursive function to print the specific pattern given in Figure 3. These questions were chosen since students are only given mathematically defined or simple list manipulation recursive functions in their course and we hoped they would give us more insight into the students’ understanding of recursion. Students were given a score based on their understanding of the active flow, limiting case and passive flow of each of the questions given in the interview. A combined score was computed for their performance in the interview, the exam question and in the other test questions, in order to determine if they fully understood recursion.

The next section describes the findings of the study.

2.2 Results of the Study

The study was able to determine trace mental models of recursion from students’ traces to the three questions. Table 1 shows the number of students and their trace mental models for each of the three trace questions the students

Algorithm1(List):
if List is empty:
    return 1
else:
    return Algorithm2(head(List))+Algorithm1(tail(List))

Algorithm2(n):
if n=0:
    return 1
else:
    return 2*Algorithm2(n-1)

Figure 1: The third test algorithm
<table>
<thead>
<tr>
<th>Question</th>
<th>Trace Mental Models</th>
<th>Copies</th>
<th>Active</th>
<th>Looping</th>
<th>Odd</th>
<th>Algebraic</th>
<th>Return-Value</th>
<th>Step</th>
<th>Passive</th>
</tr>
</thead>
<tbody>
<tr>
<td>First test question</td>
<td>viable</td>
<td>61</td>
<td>20</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>non-viable</td>
<td>7</td>
<td>10</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Second test question</td>
<td>viable</td>
<td>55</td>
<td>20</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>non-viable</td>
<td>9</td>
<td>16</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>Third test question</td>
<td>viable</td>
<td>20</td>
<td>17</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>non-viable</td>
<td>15</td>
<td>42</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1: First year students’ trace mental models of recursion for the three trace test questions, for a total of 123 students.

<table>
<thead>
<tr>
<th>Descriptive Mental Models</th>
<th>Copies</th>
<th>Active</th>
<th>Looping</th>
<th>Odd</th>
<th>Passive</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>9</td>
<td>25</td>
<td>7</td>
<td>66</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 2: First year students’ descriptive mental models of recursion, for a total of 118 students.

```python
Draw(n, dist):
    if n==0:
        return
    else:
        Draw(n-1, dist-10)
        turtle.forward (dist)
        turtle.left (90)
        turtle.forward (dist)
```

Figure 2: The algorithm given in the first interview question.

```
* *
* * *
* * * *
* * * * *
* * * * * *
```

Figure 3: The second interview question required students to design an algorithm to print this pattern.

were given in their test. Depending on the simplicity of the algorithm, more students produced a viable trace mental model. This was seen by the fact that 61 students produced a viable copies trace mental model for the first trace question but the number of students with viable trace mental models declined for the third trace question which was more complex to solve as it uses a second recursive algorithm.

It was found that mental models could not be identified from students’ proofs of the termination of Algorithm1. The reason for this is that students were required to prove that both Algorithm1 and Algorithm2 terminated for a list of non-negative numbers. Most students did not realize this. Additionally it was found that first year students have difficulty providing proofs. The question did not specify a proof technique to be used hence students attempted different proof techniques which also affected the identification of mental models from students’ solutions.

The solution to the FAC exam question required students to explain that the function would reach either one of the limiting cases then come out of the recursion in the passive flow before it terminates. It was found that all of the 127 first year students who wrote the exam said that the algorithm terminated at either of the limiting cases. These students made no reference to a backward flow of control, that occurs once the limiting case has been reached, before the algorithm comes to a complete halt. Hence students were thinking of recursion as a loop that terminates at the limiting case conditions. These results provide evidence that students do not understand the passive flow of recursion or when a recursive function terminates.

Mental models could be identified from students’ descriptions of the execution of Algorithm1. A total of 118 students answered this question. Students’ descriptive mental models were categorized based on their identification of the active flow, limiting case and passive flow of Algorithm1. If a student described all three of these cases they were said to have a copies descriptive mental model. Similarly for the other mental model types. Specifically students were said to have an odd descriptive mental model if they showed no understanding of these three cases in their descriptions. Table 2 shows the number of students and their descriptive mental models. It can be seen that only 9 students were able to correctly describe the execution of Algorithm1. A factor that affected students’ descriptions was language. Not all students are first language English speakers hence they may understand recursion but not be able to fully describe it. It was also discovered that students can obtain the active flow, limiting case and passive flow directly from the algorithm by following the pseudo-code. Students who rewrote the algorithm in words were said to have an odd descriptive mental model of recursion as they showed little insight into these three processes.

Fifteen students with viable trace mental models of recursion in all three trace questions voluntarily participated in an interview to determine if they had a comprehensive understanding of recursion. None of these students had evaluated recursive functions before their first year at Wits. It was found that only 2 students showed a complete understanding of recursion since they were able to solve the two
problems with little difficulty. Whereas 7 of the students showed a poor understanding of the active flow, limiting case and passive flow of recursion, as the interviewer had to help them solve the problems. 5 of these students had shown at least one viable copies trace mental model of recursion and 3 of these students have produced three viable copies trace mental models of recursion in their test. These results prove the assertion made by Kahney [5] that possessing a copies mental model does not necessarily imply a full understanding of recursion.

The study has revealed that most students do not understand the passive flow of recursion. This is further discussed in the next section.

3. DISCUSSION

3.1 Using Trace Methods

Tracing methods such as the boxes method [3], droids method [7] or explicitly stating the function calls of a recursive algorithm are an essential aid when teaching recursion to students [2]. They provide students with a mechanical means of following the execution of a recursive algorithm. Students can use these methods to compute the correct answer to a recursive function given the input values. Logically these methods should allow students to understand the active flow, limiting case and passive flow of a recursive function since the students are explicitly following the execution of the algorithm. George [2] showed that students can produce copies mental models when using diagrammatic traces but when asked to evaluate recursive functions without the use of traces they were unsure of their answer and felt they would need to trace the algorithm first. Although diagrammatic traces allow students to develop mental models of recursion, students heavily rely on them when solving recursive problems.

We were concerned that even though our students can produce correct traces they still do not fully understand recursion – they simply use these methods in a mechanical fashion. The results (as presented in Table 2) show that 66 students had an odd descriptive mental model of recursion. Further analysis shows that 54 of these students had at least one correct trace solution. In fact, 32 had at least one viable copies trace mental model. This confirms that many of our students can trace the active flow, limiting case and passive flow of a recursive function however they cannot describe this execution of a recursive function in words. They do not have an in-depth understanding of recursion.

It was originally decided that the interview students would not be allowed to trace the algorithms. However during the interviews students who struggled to understand the first algorithm, Draw(n,dist), were allowed to trace it. It was found that 12 of the 15 interview students had to trace the algorithm before they understood it. Nine of these students, who had to explicitly trace the algorithm, had at least one viable copies trace mental model. In fact, 6 of them had three viable copies trace mental models, one for each trace question. These students feel comfortable in explicitly tracing recursive functions and they rely on it as a means of evaluating recursive functions. They could not, however, explain what the algorithm did without doing the explicit tracing – they do not have a real feel for recursion.

These findings indicate that teaching trace methods will help students to evaluate recursive algorithms and is thus a good starting point in enabling students to understand recursion but that more is required to develop a solid understanding of recursion. The next section discusses a specific difficulty that students have and then Section 3.3 discusses some improvements that could be made when teaching recursion to students.

3.2 Understanding the Passive Flow of Recursion

A factor that affected the students’ performance in the interviews is that the algorithms given in the interview were embedded recursive functions with commands that had to be executed after the recursive call. Figure 2 shows the algorithm given to students as the first interview question which used the Python turtle module to draw an outward directed spiral. Students were asked to evaluate Draw(10, 100). The algorithm first reduced the input values until the limiting case then it came out of the recursion drawing the spiral in the passive flow, starting with the value of \( n = 1 \) and \( \text{dist} = 10 \). Ten of the interview students did not evaluate the recursive call \( \text{Draw}(n-1, \text{dist}-1) \). They first substituted the values in the function then attempted to execute the turtle drawing commands with the value \( \text{dist} = 90 \). The interviewer had to repeatedly explain to them that \( \text{Draw}(9,90) \) was another function call to the algorithm that had to be evaluated first before they could execute the remaining commands.

Additionally they had difficulty understanding the values of the variables in the passive flow. A few students thought they would be drawing with \( \text{dist} = 100 \) first, even though they reduced the values in the active flow to the limiting case. These students did not know that the algorithm suspends at the point a new function is called and returns to that point in the passive flow. This is due to the fact that these students have not seen such algorithms before. The algorithms they have seen in class do not explicitly show that the algorithm returns to the point it was suspended in the active flow in each of the functions. These algorithms usually have the same structure. The title of the function is immediately followed by the limiting case condition, which returns something. This is then followed by the else part of the algorithm which recursively calls the function, on a subset of the input. In these algorithms all they had to do was substitute the limiting case value and work out the value of the previous function calls, propagating that value back up. Even though students can effectively trace the passive flow of these functions they still do not realize that each instantiation of the algorithm goes back to the calling instantiation. Since there were no more commands to be executed, they assume the algorithm stopped at that point.

Students’ misconception of recursion in the first interview question affected their solution to the second question, which asked them to design an algorithm to print a pattern of increasing stars on a line shown in Figure 3. The solution algorithm worked similar to the \( \text{Draw}(n, \text{dist}) \) algorithm in Figure 2. It would decrease the input value \( n \) in the active flow and print \( n \text{ “ * ”} \) per line in the passive flow, starting with \( n = 1 \) until \( n = 5 \). Similar to the first question students had to understand the values that were returned in the passive flow to print the pattern. An additional factor that affected the results of this question is that first year students at Wits have not had practice designing their own recursive algorithms to solve specific problems. Hence a stu-
dent’s individual ability to program and design an algorithm affected their solution to this problem.

3.3 Implications for Teaching

Students at Wits get sufficient practice evaluating recursive functions [8]. They are taught to explicitly trace the function calls and produce a copies trace. However the results of this study have shown that this is not sufficient to teach recursion to students. The main downfall is that these students are only shown simple mathematically defined recursive problems which are easily solved by trace methods. Students should be shown a variety of recursive problems starting with the simple mathematically based functions to more complex graphical recursive functions, so that they do not develop the misconception that all recursive algorithms are similar in structure to the mathematically based algorithms.

More attention needs to be spent explaining the passive flow of recursion, as students have difficulty understanding this backward flow of control and seem to believe that the recursive function terminates at the limiting case. Evidence of this was found during the student interviews when students failed to evaluate the recursive call in algorithm \texttt{Draw}(n, \texttt{dist}). Additional proof of students’ misunderstanding of the passive flow was found in the FAC exam where it was clear that many students think recursion is similar to loops. When teaching recursion the distinction between loops and recursion must clearly be explained.

Students also require sufficient practice designing their own recursive functions. Students at Wits learn Python as their first programming language [8]. During their laboratory sessions they are given definitions of recurrence relations to implement into Python code [11, 8]. As a result many students use code or structure mapping where the student maps the conditional statements of the example onto their own recursive functions. Students at Wits learn Python as their first programming language [8]. During their laboratory sessions they are given definitions of recurrence relations to implement into Python code [11, 8]. As a result many students use code or structure mapping where the student maps the conditional statements of the example onto their own recursive functions. Students need to be shown a wide range of recursive problems, particularly embedded recursive functions with commands that have to be executed after the recursive call as these will allow students to see the return of the algorithm to the point it was suspended in the function in order to execute the remaining commands.

5. REFERENCES

[8] N. Mpoftu. FAC course notes, 2009. These notes were originally compiled by Professor Ian Sanders.