First year students' understanding of the flow of control in recursive algorithms

Ian Sanders[1] and Tamarisk Scholtz[2]
[1] University of South Africa, Pretoria; [2] University of the Witwatersrand, Johannesburg
sandeid@unisa.ac.za; tamariskscholtz@gmail.com

Abstract

Recursion is an important concept for any computer science student to master. Many first year students develop the viable *copies* mental model of recursion and can successfully trace the execution of a simple recursive function. This article discusses a study focused on determining whether the ability to successfully trace a recursive function means that the student understands recursion or whether they are simply "applying a formula". The research question investigated was thus "To what extent do students with viable trace mental models understand the flow of control of recursive algorithms?" The research followed a phenomenological approach. A group of first year students with viable mental models was identified by classifying the mental models in their answers to test questions. Fifteen of these students were interviewed. The interviews involved the students talking aloud while they tackled various tasks. Each student's understanding of the active flow, the limiting case and the passive flow was assessed. The results show that in most cases even these students have some difficulty with the active flow, are confused about the passive flow and have misconceptions about the limiting case. This implies that more careful thought needs to be given to the examples used in teaching recursion and how the concept is taught.

Keywords: Phenomenological study, recursion, mental models, flow of control.

Introduction

Recursion is an essential concept in computer science as it allows for the development of efficient and elegant solutions to many problems in the field. It is, however, widely acknowledged as a difficult concept to teach and learn and has been the subject of much research over the years (see for example Ginat & Shifroni, 1999; Levy & Lapidot, 2000; Stern & Naish, 2002; Mirolo, 2010). Typically, recursion is viewed as a process of breaking a problem down into smaller and smaller cases until the base or limiting case is reached and then building up the solution by solving the sub-problems (Wiedenbeck, 1988). A classic example of recursion is calculating the factorial of a number. Mathematically this is defined as $factorial(n) = n \times factorial(n\text{-}1)$ with $factorial(0)$ defined to have a value of 1. A recursive Python program for calculating the factorial of a given number is shown in Figure 1. For students to understand recursion, they must understand the flow of control in recursive functions. For the example in Figure 1 this flow of control would involve the process demonstrated in Figure 2 for a value of $n = 4$. The student would need to know how the problem is broken down into smaller instances of the same problem, what happens when the limiting case is reached and then how the solution is built up from the limiting case.

```
def factorial(n):
        if n == 0:
                return 1
        else:
                return n * factorial(n-1)
```

Figure 1. An example of a recursive program

```
factorial(4) = 4 * factorial(3)
factorial(3) = 3 * factorial(2)
factorial(2) = 2 * factorial(1)
factorial(1) = 1 * factorial(0)
factorial(0) = 1
factorial(1) = 1 * factorial(0) = 1 * 1 = 1
factorial(2) = 2 * factorial(1) = 2 * 1 = 2
factorial(3) = 3 * factorial(2) = 3 * 2 = 6
factorial(4) = 4 * factorial(3) = 4 * 6 = 24
```

Figure 2. The flow of control for factorial(4)

This article discusses a study which addresses the question of the extent to which first year students, who have acquired viable trace mental models, understand the flow of control of recursive algorithms.

Background

A key factor in mastering recursion requires understanding the flow of control in recursive functions but the complexity of the flow of control mechanism makes it a difficult concept for students to comprehend (George, 2000). This flow of control has two parts: the *active* flow refers to the forward passing of control where a programmer has explicitly called the function while the *passive* flow refers to the backward flow where control is automatically passed back to the function at the point where it was suspended (George, 2000; Kurland & Pea, 1985). Novice programmers need to develop a *viable* mental model of recursion in order to be able to deal with recursive algorithms (Kahney, 1989). Here a *viable model* means an understanding of recursion which will allow the student to arrive at the correct answer, either in tracing or developing a recursive algorithm. Kahney's (1989) work led to the identification of a viable mental model, the *copies* mental model, where the active flow, the limiting case and the passive flow are explicitly shown. He also identified the *looping* (or *loop*) model where recursion is seen as a form of iteration with the recursion terminating once the base case is reached (and the passive flow is ignored). The *looping* model can be viable for recursive algorithms where it is possible to evaluate the solution at the base case. Kahney (1989) also identified the *magic* or *syntactic* model where a student is able to match on syntactic elements but has no clear idea of how recursion works and the *odd* model where many misunderstandings of various types are shown. The last two models are nonviable as they do not allow the student to predict program behaviour. Dicheva and Close (1996) confirmed the existence of the *copies* model as a viable model and concur that the *loop* mental model is a result of students' misconceptions of recursion as a form of iteration. They also noted a number of nonviable mental models. George (2000) agrees that nonviable mental models are a result of idiosyncratic ideas of recursion and programming concepts.

Early work on the teaching of recursion considered animation tools to graphically show the flow of control (Wilcocks & Sanders, 1994) and thereby help the students to develop the viable *copies* mental model (Kahney, 1989). In addition, the teaching of recursion uses methods such as tracing algorithms using the "boxes inside boxes method" of Goldschlager and Lister (1982). Such methods are intended to help students develop a *copies* model. Continuous efforts have been made to determine whether first year students do, in fact, build viable mental models of recursion by studying how students trace recursive functions, analogous to the work of Kahney (1989) and Dicheva & Close (1996). Traces in specific test and examination questions over a number of years were analyzed (Götschi, Sanders & Galpin, 2003; Sanders, Galpin & Götschi, 2006). The students' mental models were then determined based on their representation of the flow of control in the recursive functions. Götschi *et al.* (2003) noted the existence of the *copies*, *loop*, *magic* and *odd* models, as identified by Kahney (1989), but identified an additional, sometimes viable, model (the *active* model) as well as additional nonviable models (*step*, *return value* and *algebraic*). In the *active* model the active flow is explicitly demonstrated but the passive flow is not shown, either because the solution has been evaluated at the base case, or because the passive flow has been implicitly dealt with. Götschi *et al.* (2003) also showed that the *looping* and *active* mental models could be considered as *risky viable models*: they can result in the correct solution for simple tail recursive functions (like factorial) but do not always lead to the correct solution. Götschi *et al.* (2003) argued that students who demonstrate the *magic* model have partial understanding of the recursive process as their traces show aspects of the active flow, limiting case and passive flow but with clear errors showing lack of understanding. The *step*, *return value* and *algebraic* models simply demonstrate various forms of confusion on the part of the student. Sanders *et al*. (2006) corroborated the findings of Götschi *et al.* (2003) as regards the first year students. Subsequent work (unpublished) identified the existence of a *risky viable passive* mental model of recursion where only the passive flow is shown.

Many first year students do, in fact, develop viable copies mental models of recursion (Götschi *et al.*, 2003; Sanders *et al.*, 2006). These students can use a tracing method (e.g. Figure 2) and successfully trace the execution of a recursive algorithm given some input values. This approach works particularly well for mathematically defined recursive functions. In this sense trace methods can be viewed as a "formula" to solve a problem since the trace methods can be applied in a mechanical fashion. A student who cannot correctly produce a trace of a recursive function clearly has very little understanding of recursion. This raises the question of whether a student who can successfully trace a recursive function has a deep understanding of recursion or whether such a student is simply mimicking the approaches used in lectures for showing how recursive functions execute. Ginat and Shifroni (1999) argue that an understanding of the basic computing model (or having a viable mental model) is not enough for students to really understand recursive formulations. They argue for a more abstract problem-level based approach. Mirolo (2010) sought to gain a better understanding of the problems that students face in mastering recursion. He devised a study to investigate whether students who had developed competency with the mechanics of the recursive process could deal with higher level skills like establishing relations in the problem domain, dealing with recursive structures and coping better with abstraction. He found that students coped quite well with the more mechanical tasks like performing recursive traces, but did not do that well in higher level tasks. This article discusses a study which addresses the matter of deeper understanding. In particular, the research focused on answering the research question:

> To what extent do students with viable trace mental models understand the flow of control in recursive algorithms?

This question can be answered by investigating the following sub-questions:

To what extent do they understand the active flow?

To what extent do they understand the passive flow?

To what extent do they understand the limiting case?

Research Methodology

An appropriate methodology to study students' cognition in this context was phenomenology (Creswell, 2009; Cohen, Manion & Morrison, 2005). Phenomenology is both an underlying philosophy and a method of data collection. It is a strategy of inquiry in which the researcher identifies the 'essence of human experiences' (Creswell, 2009: 13) regarding phenomena, and does so by obtaining information directly from participants, who describe their lived experiences. The process involves extensive engagement with a small sample of participants. The researcher then studies the expressed experiences at face value, analyses statements, and generates units of meaning to identify patterns in the experiences. These meanings are determined retrospectively by looking back reflectively at what has been occurring.

The research involved two phases: identifying students who showed a viable trace mental model and then assessing whether those students understood the flow of control. Similar to previous studies, first year students were given recursive algorithms to trace in a revision test. Their traces were categorized into the mental models of recursion defined previously (Scholtz & Sanders, 2010) and fifteen students who showed viable trace mental models for all three test questions were chosen. The second phase differed from previous research in that it had a clear phenomenological nature. Two tasks were explicitly designed to assess students' understanding, and the fifteen participants were monitored in "talk aloud" interviews as they tackled the tasks. This approach went beyond a conventional semi-structured interview, in that it implemented a phenomenological paradigm which elicited participants' spontaneous perceptions and reasoning processes during lived experiences of recursion. In line with the view of Guest, MacQueen and Namey (2012), the approaches of open-ended questions and conversational inquiry were used to support the students in articulating topics in their own words.

Talking aloud tasks

Task 1 required each student to "Describe in general what happens when a recursive function bubbles out". To be able to answer this question a student should explain that bubbling out occurs after the active flow has completed by reaching the limiting case, and the solution is built up as each instantiation returns control to the instantiation that invoked it at the point where that instantiation suspended (the passive flow). A student's response to this task can then be evaluated to determine to what extent they understand the flow of control. Task 2 dealt with recursive functions. It required the students to determine the output of a recursive Python program (Figure 3) which uses Python's turtle module. The program takes a number `n` and a distance value `dist` as input. The pattern drawn is a square spiral as shown in Figure 4 (note that the program does not produce the annotations on the figure). This program was chosen as the active flow simply reduces the problem to the limiting case, the limiting case is essentially a switch from the active to the passive flow without returning any values and the main work of the program is done in the passive flow. If a student was able to generate the correct output then they would be showing understanding of the flow of control.

Each student was asked to evaluate the given program for `n = 10` and `dist = 100`. They were given a pencil and paper which they could use to follow the flow of control of the program and draw the output. The students were familiar with Python, however they had not used the turtle module and were thus given a short introduction based on a sequential program using the same commands. This was done so as not to distract from a focus on recursion as such. The students were required to explain the limiting case, state the values of inputs at each new instantiation, and reflect aloud on the direction and distance moved by the turtle while they were working through the task.

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

Figure 3. The program given in the Task 2

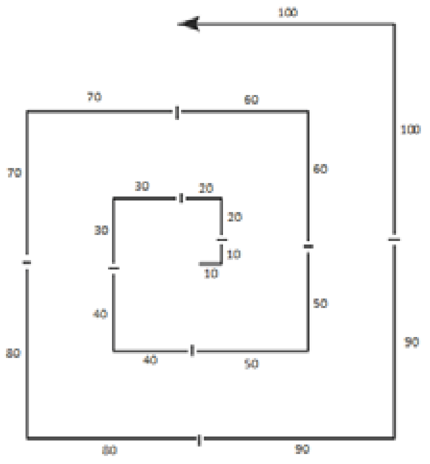

Figure 4. Solution to Task 2 for n = 10 and dist = 100

At the beginning of the interview, the student was asked about their prior use of recursion, and some general questions for demographic information. The interviewer emphasized that the interview was completely voluntary and that all information gathered would remain confidential. Students, who chose to participate, were given an informed consent form to sign. In order to assess a student's understanding, the individual talking aloud whilst solving the tasks and concurrent interviews were audio-recorded and transcribed by the interviewer. In addition, the interviewer made notes in real time during the interview.

Data analysis

The students were rated on their understanding of each aspect of the flow of control (a score from 0 to 2). In order to cope with this task, a student had to understand the active flow, limiting case and passive flow of the algorithm. To demonstrate understanding of the active flow they had to make it evident that they realised that the drawing of the spiral would not occur in the forward flow of control of the algorithm. If their verbalizations and actions indicated that they saw that the function call `Draw(n-1, dist-10)` would initiate a new recursive instantiation each time (and not do anything else), it was deemed **that they understood the active flow**. For **understanding of the limiting case**, their utterances needed to demonstrate that they realised it does not return a specific value but changes control to the passive flow. To show **understanding of the passive flow** their talk-aloud cognitive processes had to show a clear understanding of how the figure was drawn as the recursion bubbles out or unwinds.

The original intention was that students would not be allowed to explicitly trace the algorithm. The purpose of this was to determine whether they could solve the problem without using tracing methods. However there were students whose verbalized reasoning indicated that they struggled to solve the problem, and some of these asked if they could trace the algorithm. A note was made if a student traced the algorithm and this was taken into consideration when rating their understanding.

Participants' spontaneous descriptions of their cognitive experiences did not always make it clear whether they were struggling with understanding the flows of recursion or with the syntax of the programming language. Since the aim of the task was not to test whether they understood Python or the turtle module, aspects of these were explained to them if necessary. A note was made of the students who struggled with the programming concepts.

Results

Students' trace mental models

The sample of 15 students was made up of 4 females and 11 males. Three of the students had done computing in school but none had prior experience in tracing or designing recursive functions. Table 1 shows the information about the trace mental models of these students. To ensure the privacy of the participants each one has been given a pseudonym that is representative of their race and gender. Eight students showed viable copies trace mental models of recursion in all three of the test questions. Three students showed three viable active trace mental models and two showed three viable looping trace mental models. There were three students who had a mixture of viable copies, active, looping and passive trace mental models.

Table 1. Students' trace mental models of recursion for the test questions.

Pseudonym	Trace for Question 1	Trace for Question 2	Trace for Question 3
Trevor	Copies	Copies	Copies
Mark	Looping	Copies	Active
Thandi	Copies	Copies	Copies
Neeshen	Copies	Copies	Copies
Eddy	Copies	Copies	Copies
Khan	Looping	Looping	Looping
Suresh	Copies	Copies	Copies
Kyle	Copies	Copies	Copies
Jeng	Copies	Copies	Copies
Bongani	Active	Active	Active
Fatima	Active	Active	Copies
Sizwe	Looping	Looping	Looping
Zinzi	Copies	Copies	Passive
Zandile	Active	Active	Active
Thabo	Copies	Copies	Copies

Understanding of flow of control expressed in Task 1

None of the students was able to give a succinct articulation of Task 1 (see Table 2).

Table 2. Students' descriptions for Task 1.

Trevor	It means you evaluate the passive flow and back substitute your answers in
Mark	It's when the function goes back to the previous values functions, it can't calculate the values so it uses the previous functions values
Thandi	The first thing you gonna get is you gonna solve the bigger problems by solving the smaller problems first. So when it bubbles out you going to find the solution to the small problem and you will substitute it into the bigger problem to solve.
Neeshen	I think it depends on the problem. As your recursion goes, as you go through the function for a certain input the function calls on itself more and more goes up till you reach your terminating case.
Eddy	What it means is it will keep on recurring. The passive flow I think.
Khan	I think it's when it reaches the limiting case it will eventually go through different values until it reaches the limiting case.
Suresh	I think once you get to your base case and then what the passive flow and after that it's when it bubbles out

Kyle	It calls on itself. When in the function it receives itself as a function and calls itself, it basically does a loop
Jeng	It means you will eventually get to the right answer after a few steps
Bongani	You find the limiting case once you get it you are able to solve the other steps.
Fatima	I think, I think, isn't it when the algorithm basically comes to an end? You finish step through the entire function and there's no more conditions to go through.
Sizwe	I just think, when you say it bubbles out it's like, you have, cause if you work from the outside, you'll work from the outside gathering certain equations or values that like where you still have things you don't know, cause like you have your active flow.
Zinzi	That's when you finally like in an algorithm of length you take your numbers each number you go back to the first statement. You keep on taking your numbers when you reach the limiting case that's when you substitute the numbers you've got on the top statement then until you get your final answer.
Zandile	I think that it bubbles out when it has reached the limiting case and there's no other algorithms to run.
Thabo	I've forgot what this bubbling out is. ... If it reaches the limiting case, it goes back again to the passive flow...

Most verbalisations show some glimmers of understanding and some students do mention appropriate words, but clearly no one holds a true grasp of the concepts. Trevor's description seems to indicate that he knows that bubbling out is a term for the passive flow, but he does not show that he really understands how the passive flow works. His phrase "back substitute the answer in" seems to indicate a perception that values must be passed back and that these values in some combination define the solution. This is not always the case. Thandi refers to "solv[ing] bigger problems by solving smaller problems", Neeshen and Kyle refer to a function calling itself and Bongani says that once the limiting case is reached other "steps" can be solved. In addition, some students' responses do seem to indicate that they know that you build up the solution by using solutions to other instantiations (Trevor, Mark, Thandi and Zinzi). Siswe mentions the active flow. Trevor, Eddy, Suresh and Thabo refer to the passive flow. Neeshen, Khan, Zinzi, Zandile and Thabo refer to the limiting (terminating) or base case.

Understanding of flow of control expressed in Task 2

Table 3 shows the scores that each student was given for their understanding of the three phases in Task 2. Two students (Kyle, Kahn) managed to work through the algorithm relatively easily and determine what output it was going to produce. Two others (Trevor, Mark) realised quite rapidly that the algorithm would draw a spiral shape (because of the left turn commands) but assumed that drawing would be done using decreasing values of `dist` rather than increasing values. That is, they simply assumed (although they probably did not realise it) that the drawing would happen in the active flow of the algorithm. The majority of the students, however, needed quite a bit of help to understand the algorithm.

Table 3. Students' scores for the active flow, limiting case and passive flow for Task 2.

Name	Active flow	Limiting case	Passive flow	Total (out of 6)
Trevor	2	2	1	5
Mark	2	2	1	5
Thandi	1	0	1	2
Neeshen	0	1	0	1
Eddy	1	2	0	3
Khan	2	2	2	6
Suresh	1	0	0	1
Kyle	2	2	2	6
Jeng	1	1	0	2
Bongani	1	0	1	2
Fatima	1	2	1	4
Sizwe	0	2	1	3
Zinzi	1	1	0	2
Zandile	0	1	0	1
Thabo	0	0	0	0

The active flow The active flow is "active" only in the sense that the problem size is reduced to the limiting case. This clearly confused many of the students. They thought about or started drawing the figure with `dist = 100` (Thandi, Neeshen, Suresh, Jeng, Bongani, Fatima, Sizwe, Zinzi, Zandile and Thabo). Once the interviewer had corrected their misconceptions they handled this flow much better and realised that a number of recursive calls with the effect of reducing `dist` were being made.

The limiting case Eight of the students were completely baffled by the fact that the algorithm did not return a value at the limiting case (Thandi, Neeshen, Suresh, Yeng, Bongani, Fatima, Zinzi and Thabo). For example, Thandi said "But then, how are you going to evaluate the function if you not returning anything here to evaluate it with?" The interviewer had to explain the limiting case to them more than once. These students said that they were used to returning a number at the limiting case of recursive functions. Thandi, Suresh and Thabo thought that only the number 1 is returned at the limiting case of functions. Bongani thought that the algorithm would stop since it returns nothing at the limiting case. He later explained that he was used to relating the limiting case value to the other function calls. As a result he was unsure what he would have to substitute into the other function calls. Similarly Suresh said that he could not see how the algorithm would add a 1 to the answer. It had to be explained to these students that the limiting case does not necessarily have to return *something*, it provides a condition that changes the flow of control from the active flow to the passive flow and *might* also return the solution for the smallest instance of the problem.

The passive flow The students were asked to determine the shape drawn by the algorithm `Draw(n, dist)` for `n=10` and `dist=100`. The shape is only drawn in the passive flow of the algorithm after the recursive call Draw`(9,90)`has been executed. Ten students (Thandi, Neeshen, Suresh, Jeng, Bongani, Fatima, Sizwe, Zinzi, Zandile and Thabo) ignored this recursive call and wanted to start drawing the shape immediately even though they did notice that it would call Draw`(9,90)` (and they even read this out to the interviewer). The interviewer had to stop them and explain that this was a new recursive call that had to be evaluated first. Some of the students were also confused about the values of the variables as the program comes out of the recursion. Jeng had no idea that you go back to the previous values (`Draw(1, 10))`. Zandile thought that the values in the passive flow would be the initial values (`Draw(10,100))`. Others could not possibly see how the algorithm goes back to draw the figure. In particular, Suresh stated that he did not know that recursive algorithms had this backward flow. Eddy said that he ignored the passive flow.

Discussion

To what extent do students understand the active flow?

Students with copies, looping or active trace mental models should understand the active flow. The students' verbalizations in Task 1 showed very little evidence of this understanding except that some students commented on breaking down the problem into smaller instances. Only four students showed a good understanding of the active flow for Task 2. The rest of the students seemed to be confused that nothing was actually done in the active flow other than reducing the recursive calls. Limited prompting from the interviewer helped the students to deal with this aspect better but clearly they do not have deep understanding.

To what extent do students understand the limiting case?

The responses for Task 1 indicate that some students realize that recursive algorithms should have a limiting case and when this is reached the recursion stops or switches control but in general it is difficult to conclude anything else about their understanding.

The algorithm in Task 2 was different to the algorithms the students had seen before because it did not return a value at the limiting case. The algorithms they had seen would typically return a number (often 0 or 1) or an empty list (see Figures 5 and 6). The students' responses showed that they expected some value to be returned once the limiting case was reached. In fact, a number of students expected a 1 to be returned. They also expected that this returned value would be used in calculating the final solution (which is not always the case). This shows clear misconceptions about the limiting case which might return a directly calculated solution but could also simply be a switch in the flow of control. These data agree with the findings of Haberman & Averbuch (2002) that students have difficulty with identifying and handling the base or limiting case.

```
Algorithm(numlist):
        if numlist is empty:
                then return 1
        else:
                return 2 * numlist || Algorithm1(tail(numlist))

Note: Remember that || means concatenation (joining) of two lists
```

Figure 5. A simple list manipulation algorithm.

```
c(n):
        if n = 1:
                then return 1
        else:
                return 4c(n/2) + 3
```

Figure 6. A simple recurrence relation algorithm.

To what extent do students understand the passive flow?

In general, Task 1 showed little evidence of understanding of the passive flow although students with copies and passive trace mental models should have understood it.

The algorithm for Task 2 is an embedded recursive function with executable commands after the recursive call. There is an explicit suspension of an instantiation while further instantiations are dealt with before control returns to the point where it was suspended to continue executing any commands that appear after the recursive call. Prior to the interviews the students had only seen recursive algorithms where the recursive call is part of the last line of the function (see Figures 5 and 6). It is not a separate line of code followed by still more lines of code. During the passive flow of these functions students merely substituted in the values returned by the previous instantiation. Most of the students tried to use a similar approach for Task 2 but were unsure about the order in which the lines of code are executed and were confused about what values to use.

Students with copies mental models should have been able to cope with this new type of algorithm as they had shown the passive flow in their traces. This was not the case. For example, Eddy had shown three viable copies trace mental models but his verbalizations showed that he did not understand when the passive flow occurred or how it contributed to solving the problem. Active and looping mental models should have been insufficient for understanding the passive flow, but Mark and Khan showed that they understood it. The three students with active mental models, which would have been sufficient for the test questions, did not cope well with Task 2. This was to be expected as their mental models are *risky viable* and could be illustrative of misconceptions.

Extent of students' understanding

In analyzing the students' responses and articulations and in identifying patterns in their experiences, the overall findings were disappointing. It had been hoped that the students who could successfully trace recursive algorithms would have shown a deeper level of understanding but this research shows that the majority of the students (11 out of 15) could not make the conceptual jump from the correct application of rote procedures to mastery of the concept. This result is consistent with the work of Mirolo (2010) who found that students coped quite well with recursive traces but did not do so well in higher level tasks.

The four students who dealt best with Task 2 were Trevor, Mark, Khan and Kyle. Trevor and Kyle had shown copies mental models in all three test questions so it was expected that they should understand the algorithm. There were, however, other students with three copies models who could not cope with the new algorithm. Khan had shown three looping mental models so there was an expectation that he would struggle with the new algorithm. He did not. Mark had shown three different trace mental models which may indicate a deeper understanding and an ability to select an appropriate trace mental model. These data (although from a small sample) show that deeper understanding is not necessarily related to a particular trace mental model or to consistency in using one trace mental model. This result supports a hypothesis by Götschi *et al.* (2003) that students who understand the process use "appropriate" trace methods in answering test questions.

Reflections on the validity of the study

No pilot study (cf. Gall, Gall & Borg, 2010) was done as it was expected that the number of students who would be able to successfully trace all of the given questions would be small and that it would be difficult to get enough students for the study.

A concern in all interviews is that the interviewer could lead the participants and thus invalidate the data (Gall, Gall & Borg, 2010). In this study, some of the students were very weak and the interviewer had to prompt and guide them to make progress. This could be seen to be biasing the results but the students displayed a lack of understanding of recursion and the interviewer's interventions did not affect this.

As mentioned above most of the recursive algorithms seen by the first year students are either simple list manipulation algorithms or recurrence relations. The algorithm used in the interview looks very different and may have been too much of a jump for the students. An additional problem was that the students could not step away from the details of programming language syntax even though they were told that the intention was not to test their knowledge of syntax. This was compounded by that fact that Python was used but the students were being taught Pascal by the time the interviews were done.

Implications for teaching

This research on students' experiences and patterns in dealing with recursion (though conducted on a small sample) indicates strongly that the students are not developing a deeper understanding of the flow of control in recursion. Many of them can trace the execution of recursive algorithms and "get the right answer" but they do this more in a sense of following a recipe or applying a formula. They do not really understand what they are doing. Obviously being able to trace recursive algorithms is an important skill but they should master the process and need to be able to develop their own recursive solutions to problems.

Some changes have already been made in the teaching of recursive algorithms to the first years. In particular, the obvious, "more simple", recursive algorithms like factorial are not the first examples covered in class and an attempt has been made to try to introduce recursive algorithms which require understanding of both the active and passive flow (Sanders *et al*, 2006). In algorithms such as those shown above (Figures 5 and 6) the solution is built up in the passive flow. It was thought that such algorithms would be beneficial in helping the students develop a viable mental model of recursion but it seems that even those algorithms do not force the students to really think about the passive flow. They can determine the correct solution for given instances of the algorithm without really understanding the passive flow by just "dropping in" values which they can calculate. In order to teach the students to understand the passive flow better, more explicitly embedded recursive algorithms similar to the one used in Task 2 should be used early in the course. Different types of recursive algorithms and different approaches to teaching recursion should also be investigated. For example, Gordon (2006) and Stephenson (2009) argue that visual displays, as in the drawing of fractals, will assist students in understanding that computation could happen as a function goes into and comes out of recursion as well as at the limiting case. Edgington (2007) argues that recursion can be seen as task delegation which highlights the breaking down of the tasks into smaller tasks and the solution of the task using the solutions to the subtasks. Wirth (2008) uses the idea of parking cars to illustrate the concept of "divide and conquer". Approaches to teaching the students how to design their own recursive algorithms must also be investigated.

Future work

This research has shown some difficulty in coping with the active flow, some confusion about the passive flow and highlighted misconceptions at the limiting case. A further study should be done using recursive algorithms which are somewhat closer to the students' prior exposure but which allow for interrogation of the students' understanding of the limiting case and the passive flow. Other research envisaged involves studying how more senior computer science students cope with recursion in line with the work done by Ginat (2004) who showed that senior students do not always produce recursive solutions even when those would be the most appropriate solutions.

Conclusion

The present research set out to establish the extent to which first year students understand the flow of control in recursion. In an effort to do this, researchers conducted open-ended interviews with students while they were working on problems that implemented recursion. The students were required simultaneously to articulate their experiences as they worked through the tasks. Guest, MacQueen & Namey (2012) point out the value of such phenomenological research, where participants' perceptions and lived experience are the prime objects of study. The findings show that even students who have developed viable trace mental models are still confused by recursion. They have some difficulty in dealing with the active flow; are confused about what happens in the passive flow; and have misconceptions about what happens at the limiting case. In addition, the research highlighted the fact that understanding of the process is not related to the demonstrated trace mental model. Previous research has indicated that the supposedly simple tail recursive algorithms should be avoided because these can make students think of recursion as a "loop" (Sanders *et al*, 2006). This research shows that embedded recursive functions, where there are executable lines of code both before and after the recursive call, should be used as the first examples in teaching the concept.

Acknowledgements

The authors would like to thank the students who participated in the study; the editor and the reviewers for their helpful comments and Ruth de Villiers for her help in revising the article.

References

Cohen L., Manion, L., & Morrison, K. (2005). *Research Methods in Education*. London: Routledge-Falmer.

Creswell, J.W. (2009). *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. Thousand Oaks, CA: SAGE Publications Inc.

Dicheva, D., & Close, J. (1996). Mental Models of Recursion. *Journal of Educational Computing Research*, 14(1), 1-23.

Edgington, J. (2007). Teaching and viewing recursion as delegation. *Journal of Computing Sciences in Colleges*, 23, 241-246.

Gall, M. D., Gall J. P., & Borg, W. R. (2010). *Applying Educational Research: How to Read, Do, and Use Research to Solve Problems of Practice*, 6th Ed. Boston: Pearson.

George, C. (2000). EROSI: Visualising Recursion and Discovering New Errors. *Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education*, 305-309.

Ginat, D. (2004). Do Senior Students Capitalize on Recursion? *Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, Leeds, UK, 82-86.

Ginat,, D. & Shifroni, E. (1999). Teaching recursion in a procedural environment—how much should we emphasize the computing model? P*roceedings of the thirtieth SIGCSE Technical Symposium on Computer Science Education*. New Orleans, USA, 127-131.

Goldschlager,, L. & Lister, A. (1982). *Computer Science: A modern introduction*, Prentice Hall, Englewood Cliffs, NJ: Prentice Hall.

Gordon, A. (2006). Teaching recursion using recursively-generated geometric designs. *Journal of Computing in Small Colleges*, 22(1),124-130.

Götschi, T., Sanders, I., & Galpin, V. (2003). Mental Models of Recursion. *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Educatio*n, Reno, Nevada, USA, 346-350.

Guest, G., MacQueen, K.M., & Namey, E.E. (2012). *Applied Thematic Analysis*. Thousand Oaks, CA: Sage Publications.

Haberman, B., & Averbuch, H. (2002). The case of base cases: why are they so difficult to recognize? Student difficulties with recursion, *Proceedings of the 7th annual conference on Innovation and Technology in Computer Science Education*, Aarhus, Denmark, 84-88.

Kahney, H. (1989). What do Novice Programmers Know About Recursion? In E. Soloway & J. Spohrer (Eds.), *Studying the novice programmer*, (pp. 209-228). Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Kurland, D., & Pea, R. (1985). Children's Mental Models of Recursive LOGO Programs. *Journal of Educational Computing Research*, 1(2), 235-243.

Levy, D., & Lapidot, T. (2000). Recursively speaking: analyzing students' discourse of recursive phenomena, *Proceedings of the thirty-first SIGCSE Technical Symposium on Computer Science Education*, Austin, Texas, United States. 315-319.

Mirolo, C. (2010). Learning (through) recursion: a multidimensional analysis of the competences achieved by CS1 students. *Proceedings of the 15th annual SIGCSE conference on* Innovation and Technology in Computer Science Education. Bilkent, Ankara, Turkey.160-164.

Sanders, I., Galpin, V., & Götschi, T. (2006). Mental models of recursion revisited. *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, (ITICSE 2006), Bologna, Italy, 138-142.

Scholtz, T.L., & Sanders, I. (2010). Mental models of recursion: Investigating students‘ understanding of recursion. *Proceedings of the 15th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, Bilkent, Ankara, Turkey.103-107.

Stephenson, B. (2009). Using graphical examples to motivate the study of recursion. *Journal of Computing Sciences in Colleges*, 25, 42-50.

Stern, L., & Naish, L. (2002). Visual representations for recursive algorithms. *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education*. Cincinatti, Kentucky, 196-200.

Wiedenbeck, S. (1988). Learning Recursion as a Concept and as a Programming Technique. *SIGCSE Bulletin*, 20(1), 275-278.

Wilcocks, D., & Sanders, I. (1994). Animating Recursion as an Aid to Instruction. *Computers & Education*, 23(3), 221-226.

Wirth, M. (2008). Introducing recursion by parking cars. *SIGCSE Bulletin*, 40, 52-55.