Students' Mental Models of Recursion at Wits

Ian Sanders and Vashti Galpin

School of Computer Science, University of the Witwatersrand, Johannesburg, South Africa

`ian@cs.wits.ac.za, vashti@cs.wits.ac.za`

Motivation

- Recursion is a difficult concept but it is important for computer scientists to understand recursion
- At Wits we are concerned about how we can assist our students in understanding recursion [5, 2, 4]
- One thrust is to study the *mental models* (c.f. Kahney [3]) our first year students develop
- We do this by looking at the students' traces of recursive algorithms
- We did a study in 2003, 2004 and 2005 [2]
- Many students develop the *copies* model but some still develop non-viable models [2]
- In 2006 we changed from Scheme to Python as our implementation language
- Here we report the mental models our 2006 cohort of students developed

Background

Kahney describes 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" [3, p.315].

Mental models of recursion describe the students' understanding of the process of recursion.

Mental models are derived from how the *active flow*, *base or limiting case* and *passive flow* [1] are understood by the student.

Copies Model: Always viable [3]. The active flow of recursion is shown, followed by a switch from active to passive flow once the base case is reached and then the passive flow is shown explicitly.

Looping Model: Recursion is seen as a form of iteration with the recursion terminating once the base case is reached [3]. Neither the active flow nor passive flow is shown. This model is only viable for recursive algorithms where it is possible to evaluate the solution at the base case.

Active Model: Demonstrates the active flow but not the passive flow. The solution is evaluated at the base case. [2]. This model is viable in some circumstances.

Step Model: Shows that the student has no understanding of recursion, and it involves either execution of the recursive condition once, or of the recursive condition once and of the base case [2].

Return Value Model: Indicates that the student believes values to be generated by each instantiation, stored and then combined to give a solution [2].

Magic or Syntactic Model: Shows that the student has no clear idea of how recursion works, but is able to match on syntactic elements [3]. Students with this model are close to the copies model but need more exposure.

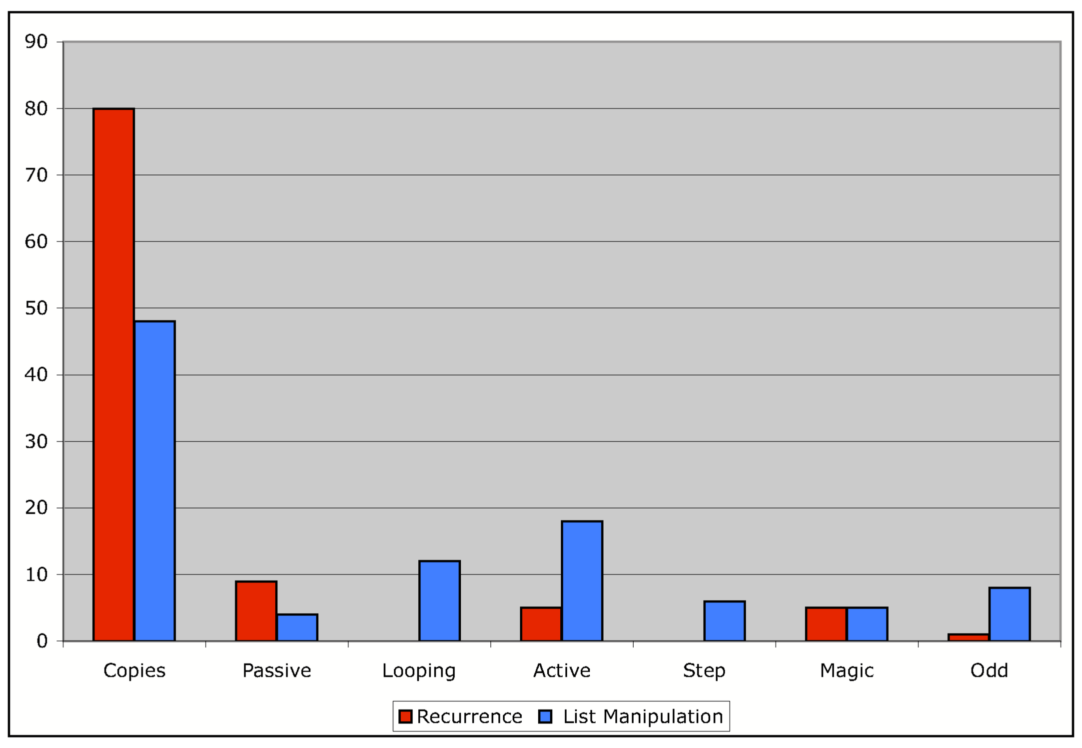
Algebraic Model: Students try to manipulate the program or algorithm as an algebraic problem [2].

Odd Model: Many misunderstandings of various types are shown. The students are not able to predict program behaviours [3].

Experiment

- Asked students to trace the execution of recursive algorithms
- Catergorised traces based on how the active and passive flow and the limiting case were shown
- Used the categorisations to describe the students' mental models

2006 Results



Conclusions

- The results are in line with our previous results
- The *copies* model is the dominant model for a recurrence relation type of recursive function but for list manipulation problems some students showed an *active* or *looping* model
- Our teaching approach, even with the switch to Python, is assisting our students in developing a viable *copies* mental model of recursion
- An interesting new result was the emergence of a *passive* mental model.
  Here the students recognised that the recursive algorithm would *somehow* get to the base case and then used the base case plus the implicit definition of the function in the algorithm to build up the required solution.

The Algorithms To Be Traced

May Class Test Question – A recurrence relation

What would the algorithm given below return as output if it was called with an input of 5. In other words, what would `TestAlg(5)` be?

```
Algorithm TestAlg(n)
 if n = 1
  then
   return 5
  else
   return 2 * TestAlg(n-1) + 3
```

Show your workings!

June Final Examination Question – List manipulation

Suppose that you are given the algorithm below. This algorithm takes as input a list of numbers (`numlist`) and returns another list of numbers as output.
Note: As in lectures, tutorials and laboratories in the course, `head(anylist)` means the first item in the list and `tail(anylist)` means the list with the first number removed, and || means joining two lists in the order they are given to make a new list.

```
Algorithm ExamAlg1(numlist)
  if numlist is empty
    then
      return a list containing the number 0
    else
      return ExamAlg1(tail(numlist)) ||
             a list containing head(numlist)/2
```

What would the output of the algorithm be if the input list was $[2, 14, 6, 12]$?
Show your workings.

Students' Traces – May Class Test

A Copies Model

$n = 5$
$TestAlg(5) = 2 * TestAlg(4) + 3$
$n = 4$
$TestAlg(4) = 2 * TestAlg(3) + 3$
$n = 3$
$TestAlg(3) = 2 * TestAlg(3) + 3$
$n = 2$
$TestAlg(2) = 2 * TestAlg(1) + 3$
$n = 1$
$TestAlg(1) = 5$
$TestAlg(2) = 2 * 5 + 3 = 13$
$TestAlg(3) = 2 * 13 + 3 = 29$
$TestAlg(4) = 2 * 29 + 3 = 61$
$TestAlg(5) = 2 * 61 + 3 = 125$

Viable – shows active flow, limiting case and passive flow

A Passive Model

$n = 1 \;\; TestAlg = 5$
$n = 2 \;\; TestAlg = 2 * 5 + 3 = 13$
$n = 3 \;\; TestAlg = 2 * 13 + 3 = 29$
$n = 4 \;\; TestAlg = 2 * 29 + 3 = 61$
$n = 5 \;\; TestAlg = 2 * 61 + 3 = 125$

Sometime viable – shows limiting case and passive flow

An Active Model

$TestAlg(5)$
$2 * TestAlg(4-1) + 3$
$2 * 2 * TestAlg(3-1) + 3 + 3$
$2 * 2 * 2 * TestAlg(2-1) + 3 + 3 + 3$
$2 * 2 * 2 * 2 * (5) + 3 + 3 + 3 + 3$
$= 80 + 4(3)$
$= 92$

Sometimes viable – shows active flow, limiting case and passive flow but has errors

A Magic Model

$n = 5 \;\; 2 * (4 + 3) = 14$
$n = 4 \;\; 2 * (3 + 3) = 12$
$n = 3 \;\; 2 * (2 + 3) = 10$
$n = 2 \;\; 2 * (1 + 3) = 8$
$n = 1 \;\; 5$

Not viable – seems to show active flow but really is just syntatic matching

An Odd Model

$2 \times (5-1) + 3 + 2 \times (4-1) + 3 +$
$\quad 2 \times (3-1) + 3 + 2 \times (2-1) + 3$
$= (2 \times 4 + 3) + (2 \times 3 + 3) + (2 \times 2 + 3) +$
$\quad (2 \times 1 + 3)$
$= 11 + 9 + 7 + 5 = 32$

Not viable – some idea of active flow

Students' Traces – June Examination

A Copies Model

$ExamAlg1(2, 14, 6, 12)$
$= ExamAlg1(14, 6, 12) \;||\; (2/2)$
$= (0, 6, 3, 7) \;||\; (1)$
$= (0, 6, 3, 7, 1)$

$ExamAlg1(14, 6, 12)$
$= ExamAlg1(6, 12) \;||\; (14/2)$
$= (0, 6, 3) \;||\; (7)$
$= (0, 6, 3, 7)$

$ExamAlg1(6, 12)$
$= ExamAlg1(12) \;||\; (6/2)$
$= (0, 6) \;||\; (3)$
$= (0, 6, 3)$

$ExamAlg1(12)$
$= ExamAlg1() \;||\; (12/2)$
$= (0) \;||\; (6)$
$= (0, 6)$

Viable – shows active flow, limiting case and passive flow

An Active Model

(1) $ExamAlg1([14, 6, 12]) + [1]$
(2) $(ExamAlg1([6, 12]) + [7]) + [1]$
(3) $(ExamAlg1([12]) + [3]) + [7] + [1]$
(4) $(ExamAlg1([]) + [6]) + [3] + [7] + [1]$
(5) $[0] + [6] + [3] + [7] + [1]$
Output would be $(0\;6\;3\;7\;1)$

Sometimes viable – shows active flow and limiting case but not passive flow

A Looping Model

1) $14; 6; 12; 2/2$
2) $6; 12; 2/2; 14/2$
3) $12; 2/2; 14/2; 6/2$
4) $2/2; 14/2; 6/2; 12/2$
$\quad 1; 7; 3; 6$

Sometimes viable – missing limiting case and incorrect passive flow

References

[1] C. George. EROSI-visualising recursion and discovering new errors. In *Proceedings of the 31st SIGCSE Technical Symposium*, pages 305–309, Austin, Texas, USA, March 2000.

[2] T. Götschi, I. D. Sanders, and V. Galpin. Mental models of recursion. In *Proceedings of the 34th SIGCSE Technical Symposium*, pages 346–352. ACM SIGCSE, ACM, February 2003.

[3] K. Kahney. What do novice programmers know about recursion? In E. Soloway and J. Spohrer, editors, *Studying the novice programmer*, pages 315–323. L. Erlbaum, Hillsdale, New Jersey, 1989.

[4] I. D. Sanders, V. C. Galpin, and T. Götschi. Mental models of recursion revisited. In *Proceedings of the Eleventh Annual Conference on Innovation and Technology in Computer Science Education*, pages 138–142, University of Bologna, Italy, 26-28 June 2006. ACM SIGCSE.

[5] D. Wilcocks and I. D. Sanders. Animating recursion as an aid to instruction. *Computers & Education*, 23(3):221–226, November 1994.