Mental Models of Recursion

Tina Götschi

A RESEARCH REPORT SUBMITTED TO THE FACULTY
OF SCIENCE, UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG, IN PARTIAL
FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

Johannesburg, 2003

Declaration

I declare that this research report is my own work. It is being submitted in partial fulfilment for the Degree of Master of Science in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other University.

Tina Götschi

_____day of _____2003

Abstract

Recursion is a fundamental concept of Computer Science that is difficult to teach and learn. Students have many misconceptions about recursion and construct mental models of recursion which are non-viable. A mental model is a student's mental representation of recursion and it provides them with predictive and explanatory powers about recursion. A mental model is non-viable if it does not allow a student to accurately and consistently represent the process of recursion.

This research analysed traces of recursive programs of first year Computer Science students at the University of the Witwatersrand. A trace was defined as a student's representation of the execution of a recursive program or algorithm. A method to identify mental models from those traces was developed. The mental models identified were the *copies*, *loop*, *odd*, *magic* and *null* models which have been previously identified, and the *algebraic*, *step*, *return value* and *active* models which are new models. The viability of each of these models and the misconceptions underlying non-viable models were examined. Suggestions of how this could inform pedagogy and curriculum were made.

The benefit of this research is that it gives lecturers a greater understanding of how students learn. Lecturers can use the method developed to identify their students' mental models and use that knowledge to provide students with examples and exercises that challenge non-viable models and thus facilitate the construction of a viable model of recursion.

Acknowledgements

First and foremost I thank my supervisors, Ian Sanders and Vashti Galpin, for all their input and expertise. Thank you for giving me enough space to construct my own knowledge but for asking the questions that challenged my misconceptions. Thanks to Sheila Rock who sacrificed one of her lecture periods, Zunaid Patel who answered Scheme questions, Ingrid Barnsley for editing, and Sangeetha Madhavan and Don Glass who answered my questions about qualitative and educational research.

I'd also like to thank Adi, Brynn, Pekka and especially my office-mate Nir, for their support either directly involved with my studies, research and writing or indirectly with my general mental health! Thanks as well too Helen, Catherine, Edan and my family for their unwavering support.

Preface

A paper based on some of the results of this research was accepted for presentation at SIGCSE 2003, the technical symposium on Computer Science Education in February 2003. This paper, titled “Mental Models of Recursion”, appears in the Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education [Götschi et al., 2003].

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

Recursion is a mathematical, problem solving and programming concept that is an important tool of a computer scientist. It is a difficult concept to teach and learn. Research examines why recursion is a difficult concept [Levy and Lapidot, 2000; Wiedenbeck, 1988], describes courses and teaching methods that have been successful [Ben-Ari and Reich, 1996; Velazquez-Iturbide, 2000; Haberman and Averbuch, 2002] and evaluates tools, such as visualisation tools [George, 2000] and algorithm animation tools [Wilcocks and Sanders, 1994], which facilitate teaching recursion. Mental models that students have of recursion have also been identified [Kahney, 1989; Bhuiyan et al., 1994; Dicheva and Close, 1996]. A mental model is the conceptual representation of an abstract concept or a physical system. This provides predictive and explanatory powers to a person trying to understand the concept or system [Wu et al., 1998].

The aim of this research was to identify the mental models that first year Computer Science students at the University of the Witwatersrand (Wits) construct and to identify the misconceptions that caused students to construct models that do not provide an accurate and consistent representation of the mechanics of recursion. This research provides lecturers with knowledge of the misconceptions that students have and the knowledge of how and why students construct non-viable mental models will benefit them in their teaching practise.

1.2 Theoretical basis

This research was concerned with identifying students' knowledge. An absolutist would define knowledge as "justified true belief" [Sober, 1995] and assume that a true

reality exists and knowledge must mirror that ontological reality. A constructivist, on the other hand, views knowledge as viable rather than true, in that knowledge allows the knower to make sense of their experiential world, rather than mirror an ontological reality. Social constructivists further believe that because knowledge construction occurs in a social context, a group of computer science students will construct knowledge that is common to that group. It can happen that a learner does not construct this common knowledge due to the influence of prior knowledge or misconceptions about processes and hence constructs non-viable knowledge.

Cognitive psychologists use the term "mental model" to define a mental representation of a concept that is derived from ontological constraints and relations in the perceived world [Johnson-Laird, 1983]. The concept of mental models has been used in the field of Computer Science Education to understand student learning [Kahney, 1989; Kurland and Pea, 1989; Bhuiyan et al., 1994; Dicheva and Close, 1996]. Wu et al. [1998, p.1] describes a mental model as "the conceptual representation of an abstract concept or a physical system that provides predictive and explanatory powers to a person in trying to understand the concept or system and guides their interaction with it". Someone with a mental model of a process has the ability to identify and mentally simulate the behaviour of that process [Kahney, 1989] or has the capacity for the mental visualisation of that process [Wilcocks and Sanders, 1994].

The basis of this research is that a student makes use of their mental model of recursion when tracing through the execution of a recursive program. A student's representation of the flow of control and the calculation of the solution of a recursive program is called their "trace". A student's conceptual representation of the abstract concept of recursion will be demonstrated by their application of that mental model in their trace.

1.3 The context of the research

This research identifies the mental models that Wits first year Computer Science students in 2000, 2001 and 2002 have constructed. A factor that influences their constructions is the curriculum of the Fundamental Algorithmic Concepts (FAC) course. The FAC course stresses the mathematical basis of recursion and its value in problem solving, but also lays emphasis on recursion as a programming technique. Thus students' knowledge of recursion will be influenced by their knowledge and understanding of programming. Learning programming is difficult and consequently students' non-viable models of recursion could develop from misconceptions about program-

ming or the programming language. At Wits, the functional language Scheme is the language used in first year. Scheme allows simple mathematical functions to be translated into program code easily and it has very simple semantics. It isolates the abstract expression of algorithms from implementation details. Thus programming in Scheme does not require much time be spent teaching a programming language or the underlying machine but rather allows lecturers to emphasise the mathematics of problem solving.

1.4 Scope and limitations

This research does not attempt to make generalisations about why students construct mental models of recursion. Rather it aims to identify Wits first year Computer Science students' mental models from 2000 to 2002 and examine these models with reference to this context. However, this does not mean that the findings of this research will not be relevant to other contexts, as Delamont and Hamilton [1984] (in [Jaworski, 1994, p.73]) say:

> "Through detailed study of one particular context it is still possible to clarify relationships, pinpoint critical processes and identify common phenomena. Later abstracted summaries and general concepts can be formulated, which may, upon further investigation be found to be germane to a wider variety of settings."

1.5 Research aims and questions

The aim of this research was to identify the mental models of recursion that Wits Computer Science students constructed and to make suggestions about the kinds of curriculum and pedagogy that may support different students in understanding recursion. The following particular questions were explored to better understand student understanding of recursion:

- Can a student's definition of recursion provide insight into their mental model?
- What viable models of recursion can be identified at Wits?
- What non-viable models of recursion can be identified at Wits?
- Do Wits students construct the same models as identified in previous research?

- Do students construct more than one mental model that they apply to different types of recursive programs?
- Does prior (iterative) programming knowledge affect construction of mental models of recursion?
- What misconceptions lead to non-viable mental models?

1.6 Research method

The choice of research methodology was influenced by the aim of identifying and understanding students' learning within the context of the Wits School of Computer Science. A qualitative research methodology, which attempts to give a holistic impression of a phenomenon [Fraenkel and Wallen, 1990], was employed.

Data was gathered from a specially developed test, from FAC examinations and from a class test. Questions that required tracing recursive programs were examined. A method to categorise students' traces was developed and these categories were used to identify mental models of recursion. The models were then examined to identify the misconceptions that could have led to these models.

1.7 Contribution of the research

The research conducted by Ben-Ari [1998] is an example of Computer Science Education research about student learning based in learning theories, but, on the whole, there is a scarcity of research based on learning theories. Often the evaluation of teaching or learning is restricted to teachers' personal experience [Carbone and Kaasbøll, 1998]. Although this research is useful, and other teachers can gain benefit from it, understanding how learning happens is valuable for evaluating students' learning and can provide teachers with specific strategies for teaching.

The analysis of students' mental models recursion that this research provides, gives teachers an understanding of how students' misconceptions can impede construction of viable knowledge. This knowledge can be used to mediate learning to facilitate the construction of a viable model of recursion.

The specific contributions to Computer Science Education that this research makes are:

- The development of a method to identify mental models of recursion.

- The identification of new mental models of recursion. Kahney [1989]; Bhuiyan et al. [1994]; Dicheva and Close [1996] have identified mental models of recursion, but constructivism leads us to expect others as it informs us that students will construct their own idiosyncratic knowledge. The new models identified are: *algebraic*, *step*, *return value* and *active* models.

- The misconceptions that can lead to the constructions of the non-viable models are identified.

- Further insight into whether students possess different mental models for different types of recursion.

1.8 Structure of document

This report is structured as follows:

Chapter 2 provides the background literature on which this research is based. The constructivist philosophy of knowledge and learning, and the concept of a "mental model" as a theory to describe the cognitive representations of knowledge are presented. These are the philosophical viewpoints of learning and knowledge that this research adopts. The difficulties that students have when learning computer programming are discussed next. The choice of language and the misconceptions about programming that students have are issues that are examined. The concept of recursion is defined. The mental models identified by previous research are discussed and the methods used by those studies to identify models are presented.

Chapter 3 discusses the method that was followed to carry out the research. The emergence of the research question from the existing research and a consideration of issues facing the Wits School of Computer Science are presented. The research aim and questions are stated and discussed. The qualitative methodology and the issues of internal and external validity and reliability are described. An overview of the process of data collection and analysis is also presented.

Chapter 4 provides a detailed description of the four stages of data collection and the analysis of each set of collected data. In each stage the questions that were addressed are described and answered and the questions that emerged from the analysis are presented. The method developed to identify mental models is de-

scribed. The mental models that were identified and discussions of misconceptions that could be the cause of non-viable models are presented.

Chapter 5 provides a discussion of the mental models identified by this research. Misconceptions underlying the construction of non-viable models are presented and some suggestions for classroom practise are made.

Chapter 6 includes a summary of the report, a revisit of the research questions and answers. There is also a discussion of the limitations of the research and some suggestions for future research. Finally, the contribution of the research is restated and an overall conclusion is made.

Appendix A contains a copy of the data collection test used in stage 1.

Appendix B contains the analysis of students' tracing methods that was not used in this research, but is included for interest.

Chapter 2

Literature Review

2.1 Introduction

The goal of teaching is to facilitate the construction of viable knowledge. This requires an understanding of how learning happens as well as an intimate knowledge of subject matter. Knowledge of a subject such as programming a recursive algorithm is not sufficient for successful teaching as knowing *how* does not mean that one will be able to teach another to do so. It can be advantageous to know what misconceptions students commonly hold and how these misconceptions lead to the construction of non-viable mental models. This research is concerned with providing teachers with this knowledge.

This chapter examines learning programming and especially recursion, describing research in these topics, which provided the background and incentive for this research. The next section describes the theory of constructivism, as this theory will provide the basis for understanding how learning takes place. Learning programming and recursion and the difficulties associated with each are then discussed. The concept "mental model" is explained, and existing research about mental models of recursion is presented. The factors that can influence the construction of mental models are considered.

2.2 The theory of Constructivism

Constructivism is a philosophical approach to knowledge and learning. Knowledge is seen to be "viable" rather than "true" as constructivism holds that the only world known, is that of our experience. This is the opposite of traditional epistemology that views knowledge as a representation of an ontological reality. Radical constructivists

believe that we cannot have knowledge of the world or consider our knowledge as certain or absolute [Von Glasersfeld, 1992] as the "real" world remains unknowable no matter how well we manage in the domain of our experience. A constructivist epistemology influences how knowledge is perceived and understanding of how learning takes place. This, in turn, influences the pedagogy believed to facilitate learning.

Constructivism rests on the hypothesis that learners actively construct their own knowledge; they do not passively receive it. To radical constructivists, learning is an active learner-centered process when knowledge is constructed and reconstructed [Von Glasersfeld, 1992]. Every learner constructs their own idiosyncratic knowledge based on their experience and existing knowledge. If experience changes, so too does knowledge. Knowledge is therefore "viable" if it fits with our real-world experiences [Jaworski, 1994]. From this radical constructivist viewpoint, it would seem that everyone has his or her own distinctive knowledge that bears no resemblance to anyone else's. By taking the role of social interaction in the construction of knowledge into account, social constructivists give an explanation for the existence of shared meaning between individuals. The mechanism by which meaning is shared is communication through language, and this results in a similar interpretation of a given concept within a group [Jaworski, 1994]. Therefore social interaction has a significant role to play in the construction of a body of shared common knowledge.

The influence of constructivism on pedagogy is considerable as the constructivist view of knowledge gives teachers insights into what their role should be to foster successful learning. Learners cannot be viewed as empty vessels that can be filled with knowledge. Teachers cannot transmit knowledge by "telling". Instead, lecturers should generate some perturbation about conceptual structures that are currently being used to attempt to foster new combinations of concepts [Von Glasersfeld, 1992].

2.2.1 The basis of constructivism

Constructivism has its roots in Piaget's genetic epistemology in which he attempted to understand how one comes to know. Piaget described knowing as tied to the action of assimilating and accommodating experiences into "action schemes" [Jaworski, 1994]. Assimilation is the integration of new knowledge into an existing structure, while accommodation describes the adaptation of existing structure of knowledge to new experiences. As Piaget puts it: "Knowledge is not a copy of reality. To know an object is to act on it" (Piaget 1967; in [Jaworski, 1994, p.15])). A new piece of knowledge cannot simply be tacked onto the existing conceptual framework of a student. There must be a growth of existing concepts of the student, by that student's own

efforts, and, in order for them to understand something new, they must have a structure that enables them to assimilate this new information.

Piaget viewed the learner as an individual and not as a cultural participant, thus disregarding the impact of the social on learning. Vygotsky's approach to teaching and learning emphasises the sociocultural, linguistic origins of conceptual thinking and the importance of communicative social interaction in learning [Jaworski, 1994]. Instruction and communication through language are thought to play major roles in shaping rational and objectively scientific modes of adult thought [Edwards, 1990]. Thus, instruction and discussion are necessary for students to construct viable knowledge. While Piaget's view of a learner has been described as a "lone organism, pitted against nature" [Jaworski, 1994], Vygotsky's theory and elaboration of the instructional relationship between a teacher and learner, acknowledges the social nature of learning and knowledge.

A question that arises, is whether learning can only take place in a social context, but this is beyond the scope of this research. As instruction at Wits does involve discussion and social interaction, it can be said that students are guided towards a common body of knowledge held by computer scientists and it is the lecturers' task to facilitate the students' constructions of this viable knowledge. The next section examines how the Piagetian theory explains the process of the constructing knowledge.

2.2.2 The process of knowledge construction

According to Piaget, the mechanisms through which existing cognitive structures are learnt are *assimilation*, *accommodation* and *equilibration* [Piaget, 1964]. When a learner is presented with new knowledge, it is either assimilated into their existing cognitive framework or, if it cannot be assimilated, causes a state of disequilibrium. In order to return to a state of equilibrium the learner's existing cognitive framework must be modified. Piaget calls this process accommodation. Piaget's notion is that the individual feels an internal drive to overcome the disequilibrium. This drive he called the "motor of cognitive development". It is the teacher's role to discover students' existing knowledge and idiosyncratic conceptions as they can serve as a basis for further knowledge construction. Non-viable knowledge will lead to disequilibrium and hence development of new cognitive structures during equilibration. During this process an individual's mental schemes usually become bigger or more complex. Thus, although the end result of learning should not be a misconception, during the learning process misconceptions can provide an opportunity for more learning [Levy and Lapidot, 2000; Piaget, 1964].

While Piaget defined mechanisms that describe cognitive development, Vygotsky's theory fails to elaborate on how learning takes place. In fact, according to Bereiter and Scardamalia [2000], no theory of learning describes the notion of learning a conceptual system more complex than the one already acquired. This problem, called the *learning paradox* is that it seems that the conceptual framework needed for a conceptual leap must be in place already. There are suggestions on how a teacher can facilitate learning but they do not describe the learning process. It is important that teachers realise that knowledge can and will be represented in different ways by different individuals. Thus students' existing knowledge plays a role in the process of constructing new knowledge and the next section looks at this in more detail.

2.2.3 Mental models

Papert [1980] wrote that "anything is easy if you can assimilate it into your collection of models". So, what an individual can learn, and how it is learnt, depends on what models that individual has available. Prior cognitive structures are needed to build new ones. The term "mental model" is used by cognitive psychologists such as Johnson-Laird to define cognitive structures or mental representations of concepts that are derived from ontological constraints and relations in the perceived world [Johnson-Laird, 1983]. The psychologist Norman defines a mental model to be an evolving mental representation of a target system formulated by learners through interaction with that target system [Norman, 1983]. Mental model research provides a level of analytic detail which observes the importance and idiosyncratic influence of the social context of learning [Gibbon, 1995]. The concept of mental models has been used in the field of Computer Science Education to understand student learning [Kahney, 1989; Kurland and Pea, 1989; Bhuiyan et al., 1994; Dicheva and Close, 1996]. Wu et al. [1998, p.1] describes a mental model as "the conceptual representation of an abstract concept or a physical system that provides predictive and explanatory powers to a person in trying to understand the concept or system and guides their interaction with it". Someone with a mental model of a process has the ability to identify and mentally simulate the behaviour of that process [Kahney, 1989] or has the capacity for the mental visualisation of that process [Wilcocks and Sanders, 1994].

In order for a mental model to be of use, that is to allow the student to make accurate predictions about the behaviour of a process, it must be consistent with a conceptual model of the process. The conceptual model is the one defined by teachers or scientists and is the shared common knowledge that provides an accurate, consistent and complete model of the process. A mental model is necessary as, without one, a

student will not be able to make any predictions about a process [Kahney, 1989]. A mental model at variance with the conceptual model will be formed as a result of a misconception about that process or any concept that underlies that process. A non-viable model should cause students' predictions of a process to conflict with experience or feedback from that process (such as a computer program) and should induce the adaptation of that non-viable model. A teacher should identify non-viable models and present students with problems that challenge those models.

Thus it is useful that teachers are aware of the misconceptions that students typically construct as it is these that lead to the construction on non-viable mental models. For novice Computer Scientists, learning programming is an area of the discipline that often leads to misconceptions. The next section elaborates on issues regarding learning to program.

2.3 Learning computer programming

While learning computer programming is not a primary objective of Computer Science courses, it is a "primary activity" or "part of the standard practices of the discipline" [Denning et al., 1988] and thus forms part of most undergraduate curricula. Universities use different methods and programming languages in their courses, but researchers agree that learning and teaching programming is difficult [du Boulay, 1989; Kay, 1996].

Five areas of difficulty that students have in learning to program are identified by du Boulay [1989] as:

1. Problems of orientation (finding out what programming is for and what problems can be solved).

2. Problems in understanding the notional machine as well as the underlying physical machine, and how these two relate to each other.

3. Problems associated with notation of the language being learnt.

4. Difficulties acquiring standard structures or plans that can be used to solve problems.

5. The issues of mastering the pragmatics of programming, that is, developing, testing and debugging programs with the tools available.

This research focuses on the fourth problem, which can be seen as the difficulties that students experience in constructing mental models of problem solving techniques.

Thus students' generation of recursive programs is not studied, but rather their interpretation or traces of programs that give an indication of their mental models. From a constructivist point of view, mental models are not constructed in isolation of existing knowledge about the computer, the programming language and problem solving in general. Therefore it is expected that students' misconceptions about these will influence their construction of mental models of recursion. This research found that often misconceptions about the computer, programming language and other problem solving techniques (such as mathematics) resulted in the construction of non-viable models of recursion.

The choice of programming language, especially a first language, is an important consideration and thus has received much attention. This matter is discussed in the next section, where the misconceptions about programming are presented.

2.3.1 The choice of programming language

A computer program, as described by Dijkstra [1989], is "an abstract symbol manipulator which can be turned into a concrete one by supplying a computer to it". According to Dijkstra a program can be viewed as a formula and it is the programmer's task to derive that formula. Thus a successful programmer requires knowledge of formal mathematics and applied logic in order to derive that formula. Secondly, a programmer needs knowledge of the types, constructs, syntax and semantics of a programming language in which problem solutions are written. The semantics of a programming language can be viewed as an elaboration of the properties and behaviour of the notional machine [du Boulay et al., 1989]. The notional machine is an abstraction of the physical machine revealed by the programming language.

Some languages have constructs and syntax which necessitate the understanding of the mechanisms of parameter passing and the control stack. To introduce these mechanisms at the beginning of a first-year course would be premature, as it requires too much knowledge about the computer [Ben-Ari, 1998; Wilcocks and Sanders, 1994]. Using a language that embodies a simple notional machine will require less prior knowledge about the computer. This allows the computer to be presented as an abstraction and algorithms can then be implemented at a high-level [du Boulay et al., 1989]. Many Computer Science educators believe that functional programming languages are ideal vehicles for instruction [Vandenberg and Wollowski, 2000] as they allow mathematically defined functions to be easily translated into program code. This mathematical approach need not be at a highly sophisticated level, thus should not be beyond the capabilities of first years. Also, functional languages do not require a deep

understanding of the mechanisms of the computer, and therefore do not require much prior study of computers.

A functional language that has gained popularity in Computer Science Education is Scheme. Scheme is a dialect of LISP created for research in artificial intelligence [Manis and Little, 1995]. It was designed to have clear and simple semantics and it isolates the abstract expression of algorithms from implementation details. It has become popular as a teaching language as it allows students to concentrate on learning good problem solving skills instead of simply language syntax and semantics or a development environment. Wits and many other universities use Scheme as the language for their introductory courses [Sanders and Mueller, 2000; Kay, 1992; Reinfelds, 1995; Vandenberg and Wollowski, 2000].

Scheme has many advantages as an introductory programming language. Owing to its simple syntax and semantics it embodies a simple notional machine allowing lecturers to teach important problem solving concepts without having to go into details about the underlying machine and simple mathematical functions can be easily translated into program code. At Wits it has been found that since it is a relatively unknown language (to students) using Scheme also has reduced the feelings of disadvantage that students without prior programming experience had in the past [Sanders and Mueller, 2000].

Although Scheme seems to be an ideal teaching language, students nevertheless construct non-viable knowledge. The next section examines why this is so.

2.3.2 Misconceptions about programming

Fleury [2000] found that misconceptions stem from the incorrect extension of a principle, from emphasising what is easily observable over what is not, or from forming rules from what is clearly understood and simply ignoring what is not. Incorrect rules are easily and often constructed by misapplying correct ones. In this way functional knowledge is extended beyond its productive range of application. Instead of constructing new knowledge through accommodation, learners simply assimilate a new concept into their existing framework. At some stage learners should experience disequilibrium and consequently modify their conceptual framework. Learners will maintain a misconception if something that challenges their mental model is not encountered and thus disequilibrium is not experienced.

Kurland and Pea [1989] found that prior natural language understanding could present problems in understanding computer programs written in English-like languages such as BASIC. Research by du Boulay et al. [1989] also found that misappli-

cation of analogies and overgeneralisation were causes of misconceptions but added that novices also struggle with programming due to inexpert handling of complexity and interaction.

Recursion is a problem-solving technique fundamental to Computer Science. It is a concept about which students have many misconceptions. Consequently, students often construct non-viable mental models of recursion. The next section describes recursion and the pedagogical implications of constructivism. Mental models of recursion that have been identified and the methods used to identify them are described.

2.4 Recursion

Recursion is an important computational problem-solving tool. A recursive solution to a problem works by breaking the problem into smaller instances of exactly the same type, solving these smaller problems and then building the solution to the whole problem from solutions of these smaller problems. For example, in order to calculate the factorial of a number, say 5, we would multiply 5 by the factorial of 4. Similarly, to calculate the factorial of 4 we would multiply 4 by the factorial of 3, and so on until we arrive at the smallest instance of the problem to which there is a solution. This smallest instance is called the base case. The base case of factorial is the factorial of 0 which is 1. An algorithm for factorial is provided in Figure 2.1 A.

George [2000] defines a recursive program as one that passes control forward actively to successive instantiations while control flows back passively from terminated instantiations. The commands that precede the recursive call are termed the *head-block* while the commands after the recursive call are termed the *tail-block* [Dicheva and Close, 1996]. A program with a recursive call between a head- and tail-block is said to have “embedded” recursion, for example Figure 2.1 B. A function could have more than one recursive call, such as calculating Fibonacci numbers as illustrated by Figure 2.1 C.

George [2000] believes that recursion is so fundamental and important to Computer Science that it should be a recurring concept in the discipline of Computer Science. A recurring concept is a concept defined in “Computing Curricula 1991” [Turner, 1991] as “basic concern throughout the discipline”. Wits lecturers agree that an understanding of recursion is essential and students have a great deal of exposure to recursion and are expected to construct viable mental models of recursion but non-viable mental models are often constructed.

Recursion is seen to be a difficult concept to teach and learn [Sooriamurthi, 2001].

`fact(n)`
`  if  n=0 then`
`    1`
`  else`
`    n * fact(n-1)`
	`prog(n)`
	`  if n= 1 then`
	`    1`
	`  else`
	`    4 * prog(n/2) + 3`
		`fib(n)`
		`  if n <= 2 then`
		`    1`
		`  else`
		`    fib(n-1) + fib(n-2)`

(A) Head-block only	(B) Embedded recursion	(C) Two recursive calls

Figure 2.1: Three types of recursive programs

George [2000] perceives that students have difficulty understanding the passive flow of control. Bhuiyan et al. [1994] agree and say that students are unable to understand the suspended computation and thus the unravelling of recursive calls. Another problem is the lack of everyday analogies [Levy and Lapidot, 2000; Bhuiyan et al., 1994]. Levy and Lapidot [2000] believe that introduction of recursion after iteration or with functions can impact on learning recursion. Haberman and Averbuch [2002] suggest that teachers should concentrate on discussing the concept of the base case as students often have difficulties identifying base cases and dealing with them when formulating recursive algorithms.

The next section discusses the process of facilitating the construction of viable mental models of recursion.

2.4.1 Constructing viable mental models of recursion

Conceptual models form the basis of methods for teaching recursion and will influence the instructional tactics used when teaching recursion. A conceptual model of recursion is one which provides an accurate and consistent representation of the mechanics of recursion. Conceptual models can be concrete or abstract [Wu et al., 1998]. Induction, the mathematical basis of recursion, is an example of an abstract model. This model is problematic if students do not have the level of mathematical sophistication required [Wilcocks and Sanders, 1994]. Another abstract model is a “structural template”. This model involves providing students with many samples of recursive programs, called “structural templates”. Similar problems can then be solved by extrapolating from the templates. This model could lead to slot-filling behaviour as students could abstract the structural features of recursion without an understanding of recursion as a problem-solving technique [Bhuiyan et al., 1991].

Wu et al. [1998] consider concrete models to be the most beneficial to novice programmers. Some examples of concrete conceptual models are:

- Russian Dolls: In this real-world analogy each doll is seen as a recursive invocation, each one of the same shape as the one before, but successively smaller, the last doll representing the base case.

- Process tracing: Sketching out the flow of control generated by recursive functions often using a graphical representation.

- Stack simulations: A recursive function or procedure is traced with reference to the sequence of activation records on the stack.

The Russian dolls model is a popular model used to introduce students to recursion [Bowman and Seagraves, 1985; Wu et al., 1998] but the analogy between the real-world and computing is very weak. Ben-Ari and Reich [1996] suggest that these simple real-world models should be strongly coupled with analogous recursive programs to make the analogy clear. They suggest the use of dramatisations to introduce recursion, the advantages being that dramatisations are fun, the explanation of the program can directly refer to the dramatisation and the visualisation of the stack is implicit and does not need to be explicitly explained. Stack simulations refer to the underlying machine and concerns detail of the computer that might be too complex to introduce in the beginning of a first year course [Ben-Ari and Reich, 1996].

Process tracing is a useful technique for teaching students about recursion. Visualising the flow of control facilitates students' understanding of recursion [Velazquez-Iturbide, 2000; George, 2000; Wilcocks and Sanders, 1994]. One such process tracing technique is the *Droid Model* [Manis and Little, 1995]. A "droid" is a diagrammatic representation of a recursive call including the name of the procedure, the arguments to the procedure and an arrow pointing back to the calling droid. This arrow explicitly represents the passive flow of control. Another diagrammatic representation of the execution of a recursion procedure uses "boxes" to represent each recursive call and show the cumulative results at the bottom or on the side of each box [Goldschlager and Lister, 1982].

Because students have different learning styles [Wu et al., 1998] it is a good idea to use a variety of conceptual models, both concrete and abstract. If software for visualising the flow of control in a recursive program is available, this can be helpful. George [2000] found that students were generally better able to evaluate recursive programs and demonstrated understanding of recursion after using EROSI (Explicit Representer Of Subprogram Invocations). Nevertheless his students still gave incorrect responses. This was owing to exogenous factors such as:

- misconceptions about variable updating and computer memory storage (e.g. variable explicitly changed by an argument),
- difficulty in evaluating conditionals, and
- idiosyncratic notions about behaviour of program elements.

A student must have viable knowledge of the programming language and its control structures as well as the notional machine that it embodies, as any misconceptions they hold about these will influence the knowledge of recursion they construct.

2.4.2 Misconceptions that prevent the construction of viable models of recursion

Misconceptions about recursion could arise from other mechanisms in imperative programming [Velazquez-Iturbide, 2000]. The problem is that understanding recursion in imperative languages also requires teaching several mechanisms like parameter passing and the control stack. The other mechanisms are complex, further adding to students' difficulty in understanding recursion. Velazquez-Iturbide argues that recursion should be approached in isolation of these mechanisms so that the concepts of only recursion can be clearly identified and isolated. He suggests a curriculum that introduces recursion in increasing levels of difficulty, first through grammars, then in functional programming and only finally in imperative programming.

Functional programming paradigms rely heavily on recursion and there is debate whether these are better suited to teaching recursion. The functional paradigm allows students to learn important concepts such as recursion. In functional programming recursion is, or should be, the only mechanism to perform repetitive computations. It encourages good programming style and problem-solving techniques without the overhead of too many types, constructs and complex syntax [Vandenberg and Wollowski, 2000]. Velazquez-Iturbide [2000] recommends that all other concepts that could arise when explaining recursion that might introduce problems must be taught first. This will diminish misconceptions of recursion owing to misconceptions of these other concepts.

Knowledge of iteration can influence learning recursion. Levenick [1990] argues that recursion should be taught before iteration. He states that iteration and recursion are interchangeable in many contexts and he perceives this to be a problem when knowledge of iteration prevents conceptualisation of a more elegant recursive solution to a problem. On the other hand Wiedenbeck [1988] found that there was not much

difference in understanding, of both mathematical problems and computer programs, whether students were exposed to recursion before iteration or vice-versa. She did find that students were better able to understand and trace through examples the more examples they had seen, no matter which repetitive technique was employed. Thus exposure to many examples (from which students can then extrapolate to understand unfamiliar problems) is beneficial [Martins, 2001].

It is important for instructors to realise that even correct examples could lead to the construction of incorrect models. For example if students' first recursive example is a simple recursive procedure with no commands in the head or tail blocks, they could construct a model that will not be viable in understanding the behaviour of an embedded recursive function. As Harvey points out (in Levenick [1990]), it is important that the instructor is aware of these shortcomings as it is unlikely that the students will be. A student who holds a misconception, has that misconception challenged and has had to construct new knowledge has had a positive learning experience. It is the teacher's role to challenge misconceptions and in order to do so, the teacher must be aware of what non-viable mental models students have.

2.4.3 Mental models of recursion

Kahney [1989] defined the *copies* model as the correct or viable model of recursion that experts possess. The copies model defines recursion "as a process that is capable of triggering new instantiations of itself, with control passing forward to successive instantiations and back from terminating ones". But as Ginat and Shifroni [1999] point out there is no evidence that this is the only model that experts possess nor is there evidence that they only possess one model of recursion. They might possess different models for different recursive tasks, such as one for designing a recursive solution and another for debugging a recursive program. Bhuiyan et al. [1994] found that learners do retain several models at one time and often possess different models for synthesising recursive solutions to explaining the correctness and control of recursive solutions.

Novices often construct non-viable models of recursion. Kahney [1989] hypothesised that novices have a "looping" model of recursion. Students with this model see recursion as some form of iteration and the recursive procedure is viewed as a single object rather than a series of instantiations. He tested this hypothesis on students after a brief training period in SOLO (an artificial intelligence language) programming and found the looping model as well as the *syntactic magic* model, where students simply recognise syntactic similarities in code and thus can make predictions about recursive

programs but have not constructed knowledge of recursion. He also found that students had various idiosyncratic models which he named *odd* models or had no model at all, which he called *null* models.

Following Kahney's work, Dicheva and Close [1996] identified mental models of recursion constructed by children programming in LOGO. The aims of this research were to identify and classify further models, to identify the misconceptions underlying the incorrect models, and to determine the possible sources of those misconceptions. With this knowledge they were able to develop a teaching approach that incorporated a diagnostic phase, where children's mental models were identified. Specific tutorial help could then be provided targeted at the non-viable model [Dicheva and Close, 1997]. Bhuiyan et al. [1994] studied novice LISP programmers' mental models, with the aim of developing a full intelligent tutoring system. *Generative* models (those that learners use when generating recursive solutions in *construction* tasks) and *trace* models (those that learners use to verify the correct solution in *interpretation* tasks) were identified. They identified different models for each type of task and have investigated how to build effective support tools for learning based on a learner's mental model.

In the empirical study conducted by Bhuiyan et al. [1991] the "loop", "syntactic", "analytic" and "analysis/synthesis" mental models were identified. These were *generative* models, meaning that students used them to generate recursive programs.

- The loop model is the flawed model of recursion that novices often construct, as Kahney hypothesised.

- The syntactic model is similar to Kahney's. Students who have little conceptualisation of recursion as a problem-solving technique, but have considerable declarative knowledge about recursion construct it. Students have recursion *templates* with empty slots for aspects such as the base case and the recursive call. The students simply fill in these slots when generating a solution.

- Students with the analytic model view recursion as a problem solving technique and not a physical construct as those with the syntactic model do. These students generate a recursive program by analysing input cases, plan a high-level solution and then implement that solution in a programming language.

- The most powerful model is the analysis/synthesis model. It is the model that experts possess and it gives them the ability to reduce a given problem into smaller ones and to synthesise the corresponding solutions into a global solution for the problem.

They named the models that students use to verify the correctness of their recursive solutions *trace methods*[1]. Trace methods allow students to debug a recursive program they have generated and gives an indication of the mental representations that students have of the abstract machine. The trace methods found were

- The stack method, where the students see each successive recursive call as new environment.
- The staircase method, where the student draws an analogy between climbing a staircase. Every recursive call is one rung up or down.
- The tree method, where the student views the solution as a tree of partial results.

Bhuiyan et al. [1991] found that a student's existing mental model is enhanced as new material is learnt. Thus students adopt more sophisticated mental models as the need arises. But they also found that once mental models are formed they are persistent, as students frequently return to more established mental models when a first attempt to solve a problem with a new model fails. They found that students construct viable models of recursion as they are presented with more difficult recursive problems because students will only apply new mental models when their established mental model is not working. They used the understanding of student learning, gained from their study, to build effective support tools for learning recursion [Bhuiyan et al., 1994], such as the PETAL system, a learning environment that helps students in understanding how to write recursive programs in LISP and assists students in the use of viable mental models.

Dicheva and Close [1996] also studied generative and trace models. They identified the correct "copies" trace model and the incorrect "loop", "do-it-once-more" and "simple" models. These are discussed in the following section.

2.4.4 Identifying mental models of recursion

The methods used to identify the mental models that students possess, require that students either interpret (or trace) the execution of a given recursive program or, given a problem definition, develop a recursive program that provides a solution.

Kahney [1989] began with the hypothesis that novice programmers have a looping model of recursion while experts possess a copies model. To confirm this he developed three SOLO programs and determined the solution of each based on his hypothesised

[1] They use the term *method* rather than *model* to emphasise how the user constructs his or her model of the system using their mental representation of that system

```
TO PATTERN3 : W
 IF EMPTY? : W [STOP]
 PRINT: W
 PATTERN3 BF : W
 PRINT : W
END
```

```
LEGO
LEG
LE
L
L
LE
LEG
LEGO
```

Figure 2.2: Program and correct solution indicating a copies model [Dicheva and Close, 1996]

looping and copies model. He then asked respondents which program would achieve the results that were required. The respondents were 30 novices with brief training in SOLO (their programming experience was not known) and 9 experts (with previous experience in LISP and PASCAL). The results recorded the program, or programs, that each respondent selected as the correct solution as well as their discussion of why they thought a program would not achieve the required result. The results showed that most novices have a looping model and rarely have a copies model, while all of the experts had a copies model. Further models were also identified: the syntactic magic model, odd models and the null model.

Subsequent research has identified other non-viable models of recursion but no other viable ones. Kurland and Pea [1989] interviewed children with a year of LOGO experience. They were asked to think aloud and predict, with a hand-simulation or trace, what LOGO programs would do. After analysis they found that children possessed a non-viable looping model.

Dicheva and Close [1996] also studied children programming in LOGO. They developed LOGO programs, with embedded recursion, that produced output and interviewed children as they interpreted (traced) the execution of those programs. One example program and output is provided in Figure 2.2.

They began the study with Kahney's copies and looping models but no other predefined models. New mental models were identified from the format of incorrect output. Figure 2.3 shows the different incorrect solutions children obtained and the mental model that was identified from each.

The loop models share the similarity that the recursive call modifies the value of the parameters and the procedure is re-invoked, but the base case causes the recursion

LEGO EG0 GO O	LEGO EG0 GO O O	LEGO EG0 GO O LEGO	LEGO EG0 GO O LEGO EG0 GO O	LEGO EGO EGO	LEGO EGO LEGO	LEGO EGO
Truncated Procedure Loop Model	Proper Loop	Embedded Loop	Two Loops	Proper Do-it-once-more	Embedded Do-it-once-more	Simple Model

Figure 2.3: Incorrect solutions and mental models

to end. The different loop models stem from the manner in which the head-block (the commands before the recursive call) and the tail-block (commands after the recursive call) are executed.

- Truncated loop: the head-block is executed as a loop, the tail-block is not executed at all.
- Proper loop: The head-block is executed as a loop and the tail-block is executed once.
- Embedded loop: The head-block is executed as a loop and the tail-block is executed once with the initial variable value.
- Two loops: both the head-block and tail-block are executed as loops.

Children with a "do-it-once-more" model, execute the head-block once more after the recursive call. The recursive call is not executed again; then the tail-block is executed.

- Proper do-it-once-more: head-block is executed twice and the tail block once (variables have the last assigned values).
- Embedded do-it-once-more: head-block is executed twice and the tail block once (variables have the initial values).

Finally those with the simple model have no understanding of the successive calls to the procedure and the inputs are simply evaluated and printed.

In further work, the misconceptions that underlie each of these models, were identified [Dicheva and Close, 1997].

- Truncated loop: The LOGO command STOP, stops the execution of an embedded recursive procedure. STOP stops all currently active processes. A recursive call is a looping construct and the head-block is the loop body.
- Proper loop: A recursive call is a looping construct and the head-block is the loop body. In a LOGO program there is only one variable, with a given name, created. So children believe that if the same procedure is called, it should use the same variables and the computer knows what values to assign to the variables.
- Embedded loop: Again, children believe that a recursive call is a looping construct but also that LOGO *remembers* the initial value of a variable for later use in the tail-block.
- Two loop: children treat the head-block and tail-block as a units, each with the status of a procedure. The head-block is executed as a loop and then the tail-block. The computer remembers the initial value so when the tail block is executed it will begin with the same initial value.
- Proper do-it-once-more: The recursive call is a command to repeat the head-block once more.
- Embedded do-it-once-more: The STOP command does not STOP execution but is seen as part of an IF statement which checks input values. The recursive call is a command to repeat the head-block once more. The computer remembers the initial value so when the tail block is executed it will use this initial value.
- Simple: The recursive call evaluates its input and simply executes the procedure statements.

Thus mental model research is in agreement that the copies model is the viable model. The non-viable looping model is also always found no matter what the language of instruction or who the subjects being studied are. Other models seem to be dependent on the language and the specific recursive programs used in the study.

2.5 Concluding remarks

Constructivism presents a view of knowledge and the acquisition of knowledge that provides a justification for learners' knowledge constructions and gives teachers in-

sight into why constructions may be non-viable. The theory of constructivism offers an educational researcher the background necessary to evaluate learning and teaching and to recommend practices that can enhance both.

In Computer Science education research, contributing to an understanding of how students learn programming and recursion can be enhanced by constructivism. There are many issues that impact on students' construction of knowledge of recursion and it would be of use to lecturers to become more aware of how matters such as programming languages, conceptual models and prior experience influence the construction of mental models of recursion. As mental models are the manifestation of a student's knowledge of recursion, research identifying viable models is of interest and research identifying non-viable models can benefit educators.

This chapter has provided the basis from which this research has developed. The main concern is what mental models of recursion students construct, how those models can be identified and what misconceptions cause the construction of non-viable models. The next chapter discusses the methodology that the research employed in order to answer the research questions.

Chapter 3

Research Method

3.1 Introduction

The previous chapter provided an account of a theory of learning, constructivism, and a theory of mental representations of knowledge, mental models, that provides the theoretical basis for understanding how learning happens and how cognitive representations of knowledge are understood in this research. It also contained a discussion of previous research in the field of student learning, specifically regarding understanding student learning of recursion and identifying students' mental models of recursion.

The aim of this study was to gain a greater understanding of what Wits first year students learn about recursion from the Fundamental Algorithmic Concepts (FAC) course and to inform the kinds of curriculum and pedagogy that may support different learners in understanding recursion. This chapter describes the research method that was used to achieve this aim and to answer the questions that were posed in order to understand student learning.

This chapter begins with a description of the context in which the research took place, namely the Wits School of Computer Science, and how concerns and issues that its lecturers face, inspired the research. The specific research questions are listed and a motivation for each is given. The qualitative research methodology is presented and its appropriateness to this research is discussed. The issues of reliability and validity, central to the soundness of the research, are examined with reference to qualitative research. This is followed by an outline of the implementation of the research, which is then discussed in detail in Chapter 4.

3.2 Research context

At the School of Computer Science at the University of the Witwatersrand changes were made to the first year curriculum in the early 1990s and again in 1998 [Sanders and Mueller, 2000]. Since 1999, Scheme has been the language of programming instruction. This was the first time a functional language was used in an introductory Computer Science course at Wits. The changes to the course and the choice of Scheme stemmed from the following concerns [Sanders and Mueller, 2000]:

- The perception of students who do not have prior programming experience that they are at a disadvantage.
- The desire to base problem solving and the study of algorithms on sound mathematical principles and have a close link between the (mathematical) statement of a problem solution and program code.
- The need to dispel the misconception that studying Computer Science merely involves learning to program.
- The need to make the course accessible to any students with the required mathematical ability, but not necessarily programming skills.
- The belief that a university course should provide an education in the discipline of Computer Science and not training in programming.

The conclusion of lecturers and students is that these changes have been successful. These conclusions are based on questionnaires that students have completed about their perceptions of the course [Galpin, 2001] and from examination results [Sanders and Mueller, 2000]. However, students continue to struggle to understand recursive programs or write ones of their own. They hold misconceptions, which result in the construction of non-viable mental models of recursion. The aim of this research is to gain a greater understanding of the process of students' construction of knowledge of recursion, and to inform the kinds of curriculum that may support students in understanding recursion. Recursion was chosen as the specific concept to study since proficiency in programming in Scheme requires a viable mental model of recursion as many of the programs students are presented with or asked to design involve recursion.

3.3 Research aim and questions

The aim of this research is to identify the mental models of recursion that Wits Computer Science students have constructed and to make suggestions about the kinds of curriculum and pedagogy that may support different students in understanding recursion. The following particular questions were explored to better understand student understanding of recursion:

Can a student's definition of recursion provide insight into their mental model?

Students often use different language to experts. Identifying the language that students use to describe recursion can provide insights into students' thinking [Levy and Lapidot, 2000], so students' language could provide a means for identifying their mental model of recursion.

What viable mental models of recursion can be identified?

Kahney [1989] defined the *copies* model as a viable mental model of recursion, but this may not be the only one [Ginat and Shifroni, 1999]. Identifying other viable models will provide a greater understanding of what knowledge students construct of recursion.

What non-viable mental models of recursion can be identified?

A variety of non-viable mental models have been identified in previous studies [Kahney, 1989; Kurland and Pea, 1989; Bhuiyan et al., 1991; Dicheva and Close, 1996]. From a constructivist viewpoint, learning involves each individual making sense of their own experiential world and thus we should expect other non-viable models of recursion. Identifying the non-viable models that Wits students construct and the misconceptions on which they are based, provided lecturers with insight into students' learning. This knowledge can inform lecturers of the kinds of curriculum that may support different learners in understanding recursion.

Do Wits students construct the mental models identified in previous research?

Most previous studies have identified the copies and looping mental models defined by Kahney [1989]. Kahney hypothesised that these are the models that experts and

novices have respectively. These models have also been identified in other research (Kurland and Pea [1989]; Bhuiyan et al. [1991]; Dicheva and Close [1996]), but unique non-viable models have been identified. Each of these studies targeted vastly different subjects (children, university students, computer science students, psychology students) and used different languages (SOLO, LOGO, LISP) and their own unique programs or algorithms. Thus it would seem that the context, language and particular programs that students are asked to interpret or generate, would influence what mental models are constructed.

Do students possess more than one mental model that they apply to different types of recursive programs?

Ginat and Shifroni [1999] have questioned whether students possess different mental models for different types of recursion. This is not unlikely since, according to constructivism, students construct knowledge that allows them to make sense of their experiences. If students experience recursive programs, with a mixture of characteristics, differently, we would expect them to construct different mental models.

Does prior (iterative) programming knowledge affect construction of mental models of recursion?

One influence on the construction of mental models could be prior, iterative programming knowledge. While Levenick [1990] believes that recursion should be taught before iteration, both Wiedenbeck [1988] and Martins [2001] found that there was no significant difference in understanding of both mathematical problems and computer programs, whether students were exposed to recursion before iteration or vice-versa.

What misconceptions lead to non-viable mental models?

A misconception is the source of a non-viable model. Identifying the misconceptions that lead to the construction of non-viable mental models would inform lecturers of the kinds of curriculum and pedagogy that may support different learners in constructing viable mental models of recursion.

These questions provide input into what kind of data and how data should be collected. They also give emphasis to the exploratory nature of this research. Due to its exploratory nature, qualitative methodology was chosen to be the most appropriate research methodology. The following section discusses qualitative methodology in detail.

3.4 Research methodology

This research did not begin with a hypothesis of what mental models students construct, but had the intention of identifying mental models that students have constructed and using this knowledge to inform curriculum and pedagogy. Qualitative research attempts to get a holistic impression of a phenomenon [Fraenkel and Wallen, 1990] and assumes that realities are socially constructed through individual and collective definitions of a situation [McMillan and Schumacher, 2001]. Ben-Ari [1998] maintains that a researcher working from a constructivist viewpoint should use qualitative methods as the underpinnings of qualitative research are essentially constructivist.

The criteria for judging the soundness of any research are its reliability and external and internal validity. Reliability refers to the consistency of data collection instruments. External validity is the degree to which the results can be generalised and transferred to other contexts. Internal validity is the degree to which the explanation of phenomena matches the realities of the world. Since, qualitative philosophy rejects the assumption that there is a reality external to our perception of it, these criteria need some re-interpretation in the case of qualitative research. The following sections provide a definition of these terms as they are understood in this research.

Internal validity

Internal validity involves establishing that the results of research are credible or believable. It important to consider other factors that could have influenced the results [McMillan and Schumacher, 2001] and how any bias, from either the researcher or the research instrument, could have affected the research. Qualitative research cannot be objective so others should corroborate interpretations made. Negative cases and discrepant data must be actively searched for and recorded. It is important to seek alternative explanations and always acknowledge these as they happen [McMillan and Schumacher, 2001].

Reliability

A research instrument is reliable if it consistently produces the same results, but in qualitative research the same thing cannot be measured twice. If we measure (or observe) something again, we are measuring two different things. It is important to give an account of the context of qualitative research and how changes in the context affected the research. Another problem is that the results represent the researcher's own interpretation of the phenomena. It is important to convince the reader that both the

research instrument and method are reliable. This will provide the justification that the conclusions that this research makes, follow logically from the analysis of the data. Strategies that ensure reliability include the use of verbatim accounts and literal descriptors, and the use of important terms that are understood by both participants and the researcher. Reliability can be maintained through reflection and alternative explanations for phenomena should be sought [McMillan and Schumacher, 2001]. Experts can be consulted for their interpretations and opinions. Reliability can also be maintained by actively searching for, recording and analysing negative cases or discrepant data [Jaworski, 1994].

External validity

External validity or transferability refers to the degree to which the results of qualitative research can be generalised or transferred to other contexts or settings. The qualitative researcher can enhance transferability by doing a thorough job of describing the research context and the assumptions that were central to the research. The person who wishes to "transfer" the results to a different context is then responsible for making the judgment of how sensible the transfer is [Trochim, 2000].

3.5 Research process

The process of the research was circular, with data collection and analysis providing feedback that refined and expanded the research questions. This brought about more data collection and further analysis. The research questions, data sources and method of analysis were not fully determined before the research took place, but the aim of identifying mental models of recursion provided the focus for the research process. The following sections provide an overview of the data collection and analysis and describe how validity and reliability were maintained.

3.5.1 Data collection and analysis

Data that would provide insight into students mental models was necessary. In order to indicate a student's mental model the data had to give an indication of a student's thought processes. There were approximately 180 students in first year in 2000, 2001 and 2002. Each student attended one of three laboratory sessions per week. In lab sessions students worked on programming assignments incorporating all aspects of programming and problem solving, not only recursive problems and programs. Thus

it would have been difficult to gather data about students' mental models of recursion using participant observation, a data collection method often used in qualitative research. What was necessary was a more directed data gathering method, one that would target as many students as possible and gather specific data only about mental models of recursion in a short period of time.

In order to collect this specific data, a test was developed. This test asked students to list their prior programming experience, to give a definition of recursion, and to trace the execution of 4 recursive programs (see Appendix A). The instruction to trace was explained as follows: *Please show as many of your thought processes as possible, either by using diagrams to represent what you understand to be happening or by writing English statements*. Thus a student's representation of the flow of control and the calculation of the solution of a recursive program is called their "trace".

Traces were analysed to identify characteristics that would provide an indication of their mental model and significant issues that warranted further investigation. This further investigation was to be in the form of interviews. At the outset of the research, the test data and interview data were to be the only data sources. However the test data was not sufficient and volunteers for interviews were not forthcoming. The data from the few that were interviewed did not provide additional insight. Thus this data was not adequate for identifying mental models and another source of students' traces of recursive programs was needed. Other traces, from the June 2000 and 2001 FAC exams, were analysed. A more formal analysis method was developed that involved coding each trace into categories that defined certain characteristics of the trace. Once traces were coded into these categories, a method to identify mental models could be developed. Further data was gathered by incorporating both the 2000 and 2001 exam questions in a class test given to the 2002 first years.

3.5.2 Internal validity

As the administration of the test was a non-interfering data collection strategy, the influence on how respondents interpreted the instruction and traced the programs was limited. The open-ended question in the first test allowed the participants to answer in their own words and recursive trace questions allowed students to use a tracing technique of their choice. A variety of recursive programs were used and the programs used simple Scheme control structures to minimise the errors due to misconceptions about Scheme syntax. Nevertheless the specific programs used still posed a threat to internal validity, as students would be influenced by the instruction and on the specific program and programming language used. The questions and programs used

are included in this report and interpretations are made explicit. Data was examined carefully to differentiate between Scheme-syntax misconceptions and misconceptions about recursion.

Bias would have been introduced in the interview data, as interviewees were volunteers. Volunteers tend to be more intelligent than non volunteers [McMillan and Schumacher, 2001]. This characteristic of volunteers made interview data unsuitable for this research. Those students who volunteered for interviews where those with viable models of recursion, thus could not add to understanding of the misconceptions that lead to non-viable models.

Although the questions in the exams were not specifically developed for this research, they allowed the research questions to be answered. They are similar to the recursive programs students are taught in lectures and tutorials and expected to trace and implement in tutorials and laboratory sessions. Thus they are the types of questions that students are familiar with and construct their mental models from. Nevertheless one limitation of this research was the lack of control over these questions. This is discussed in detail in section 6.4.

3.5.3 Reliability

In order to maintain reliability there needed to be evidence to show that attributes of significance and interpretations which were made were reasonable and of value [Jaworski, 1994]. This was done during each stage by making clear all interpretations and searching for, recording and analysing negative cases or discrepant data. All results were discussed with 3 experts. These experts were the course lecturer, another lecturer from the School of Computer Science and a course tutor. These 3 experts confirmed interpretations, offered alternative interpretations and raised questions that warranted further investigation.

3.5.4 External validity

The method developed for identifying mental models can be used in other contexts, although the specific recursive programs used to identify models may have to change depending on the programming language and types of recursive programs with which students are familiar. Thus mental models identified in a different context may be different to those found by this research. Every new mental model that is identified will have to be examined in detail in order to understand the misconceptions that led to its construction. Thus the findings of this research are not directly applicable to any

other context; in fact they are not even applicable to the 2003 Wits first year class. On the other hand, the knowledge that this research has generated can be applied by any lecturer teaching recursion and thus the research is of value to Computer Science Educators in general. To re-iterate the passage quoted in section 1.4:

> "Through detailed study of one particular context it is still possible to clarify relationships, pinpoint critical processes and identify common phenomena. Later abstracted summaries and general concepts can be formulated, which may, upon further investigation be found to be germane to a wider variety of settings."

Delamont and Hamilton [1984] in Jaworski [1994, p.73]

3.6 Concluding remarks

This chapter has described how the initiative for this research emerged from study of existing research and an interest in identifying the mental models of recursion that Wits first year Computer Science students construct. The process of the research involved data collection and analysis and the development of a method to identify mental models of recursion from students' traces of recursive programs. The following chapter describes this process in detail, making clear all decisions made during the research and presenting the findings of the research as they emerged from the analysis.

Chapter 4

Data Collection and Analysis

4.1 Introduction

The previous chapter gave reasons for the choice of research questions, which emerged from a study of the existing research and the context in which the research took place. The research methodology was presented and the issues of reliability and validity were considered both generally, as issues of concern in any qualitative research study and specifically, how they were ensured in this study.

This chapter provides a detailed description of the implementation of the process of this research. While the next section provides an overview of the process, the rest of this chapter describes each stage of data collection and analysis in detail. The results of questions addressed in each stage are discussed and the significant observations that emerged and led to consequent stages are presented. The outcome of the research, that is, the mental models of recursion that were identified, are presented in this chapter but discussed in detail in Chapter 5.

4.2 Overview

Four sets of data were collected. At each stage, data was analysed and questions that led to more data collection were raised. Thus the process of data collection and analysis was circular. The analysis of the first data began informally with the recording of significant findings. A feature of a trace was characterised as significant if it was directly related to a specific research question or was completely unexpected. These findings provided input for the second stage of data collection. Firstly they provided questions that were to be asked in interviews and, when interviews were not fruitful, gave input into what type of data would be suitable for analysis. During the analysis

of the second set of data a process of coding data was developed. Each item of data was coded according to what aspects of the definition of recursion it exhibited. This coding process was applied to the third set of data and a method to identify a students' mental model was developed. The final set of data was categorised directly into mental models.

The data sources are listed below (the stages were in chronological order although the data sources were not):

- Stage 1 : Test and questionnaire (prior experience, definition of recursion, 4 traces) administered October 2001
- Stage 2 : One recursive-trace question from June 2001 exam (this became the second question in stage 4 so is called Q2)
- Stage 3 : One recursive-trace question from June 2000 exam (this became the first question in stage 4 so is called Q1)
- Stage 4 : Class test, two recursive-trace questions (Q1 from 2000 examination and Q2 from 2001 examination) administered in May 2002

Although the June 2002 FAC exam also had a recursive-trace question that could have been included in this study, it was decided not to include this data. A method for identifying mental models had been developed and mental models had been identified with the data collected from the four stages. To ensure the timely completion of the research, no further data was collected nor analysed.

4.3 Stage 1

A method used in previous research to identify mental models of recursion, involved developing specific recursive programs that produced screen output. Students' mental models were identified from the output they produced of their hand-simulations of the programs [Bhuiyan et al., 1994; Dicheva and Close, 1996, 1997]. Another method used to identify mental models was to give students a range of programs, some that would and some that would not achieve what was required, and ask students to indicate which programs achieve the desired results [Kahney, 1989]. The first method requires programs that produce textual or graphic output. The second requires a hypothesis of the non-viable models of recursion that students possess in order to develop programs that will test that hypothesis.

The recursive programs that are used in the FAC course do not contain output commands, thus identifying models from program output would not be feasible. Also, this research did not begin with a hypothesis of the models that students possess. It was decided to identify students' models from their representation of the flow of control of a recursive program as it executes. This was termed a "trace". Although students should trace every recursive program in the same way (because the flow of control is always the same), this does not often occur. Students traced programs differently depending on each program's unique aspects, and their own misconceptions.

Four different recursive programs were developed and put in a test. The test also contained a section where students could fill in their prior programming experience and a question asking students to describe recursion in their own words. This test was given to volunteers in the 2001 first year class. A full copy of the test can be found in Appendix A.

The four recursive programs were:

1. A function that manipulates numbers, but does not calculate any known, useful function. This function contained no commands in the head block.

2. A list manipulation program that returned "true" if a specified element appeared twice in a given list. This was a fairly complex program as it used another function called `helper`. The recursion itself was in `helper` and had no head-block commands.

3. A program calculating the *n*th Fibonacci number, with two recursive calls.

4. A list manipulation program with embedded recursion. This function built a new, more complicated list, out of the given list. It contained commands in both the head-block and tail-block.

The only control structures (besides recursion) that the programs contained were the `if` statement and the `car`, `cdr`, `cons` and `null?` functions. The data the programs manipulated were numbers or simple lists. This simplicity was to ensure that difficulties students had with understanding the program behaviour stemmed from difficulties in understanding recursion and not Scheme syntax. The variety of programs would show whether students construct different models for different types of recursive problems. The specific programs were chosen for the following reasons:

- Programs with a mix of head-block and tail-block commands to investigate whether students understand both the active and passive flows of control in a recursive program.

- Useful programs (e.g. Fibonacci) and nonsense programs to investigate whether students have a *syntactic magic* model of recursion. That is, if they could accurately predict real program behaviour but not nonsense program behaviour, then it could be concluded that students were merely inferring behaviour from what the program seems to be doing and from some sort of "magic" understanding of recursion.

- Two recursive calls to further investigate their understanding of the flow of control of recursive procedures.

The test was administered in a lecture period where 54 students of the 55 students present participated. This represented less than half of the first year class. Unfortunately only sixteen tests were completed fully (i.e. a trace and solution provided for all four programs). Questions were left out owing to personal choice or time constraints. This was unfortunate as it meant that comparisons between students' mental models for different types of recursive programs could not be made. Also, many students did not attempt to give a definition of recursion nor describe what each program did.

4.3.1 Analysis of test data

The tests were analysed and the following features recorded: prior programming experience, correctness, definition of recursion, tracing method, description of programs. These five features and significant findings made are discussed next.

Prior programming experience and correctness of traces

Twenty-two of the 54 respondents had prior programming experience in one or more of the following categories:

- Pascal, studied at school.

- Java, studied at university.

- Another language (C, C++, Java, Delphi, Visual Basic), self-taught.

Thus all prior experience was experience of an imperative language. Table 4.1 summarises the findings. At this stage mental models had not yet been identified so "correct" means the answer obtained was correct and does not indicate mental models.

Table 4.1 shows that prior (imperative) programming experience does not greatly influence a student's ability to reach the correct solution. This is consistent with previous findings [Wiedenbeck, 1988; Martins, 2001]. Although more students without

		Prior Experience	No Experience
Program 1	correct	77%	72%
	incorrect	18%	28 %
	incomplete	5%	0%
Program 2	correct	27%	25%
	incorrect	41%	44 %
	incomplete	32%	31%
Program 3	correct	45%	62%
	incorrect	36%	19%
	incomplete	19%	19%
Program 4	correct	36%	25%
	incorrect	23%	19%
	incomplete	41%	56%

Table 4.1: Program correctness by prior programming experience – 2001 Test

experience obtained the correct solution for program 3, it was found that both experienced and inexperienced students made the same types of mistakes. Reasons for other differences are discussed in later sections.

Definitions of recursion

There was a wide variety of definitions of recursion and it was evident that students struggled to express themselves clearly, as the following definition illustrates:

> "It's like starting with an egg, and you get a chicken, which will bear an egg, which bears another chicken till the chicken or the egg dies. It calls itself."

Research exploring students learning found that students had difficulties in describing their engagement in learning tasks, due to their lack of vocabulary needed to do so [Carbone and Hurst, 1999]. So, it was expected that students who were not first language English speakers would have difficulties expressing their thinking, although it seems that students, in general, do not have the skills needed to clearly express their thinking. This could be because they are not often required to do so and fail to develop this skill.

Recursion has an active flow where control is passed forward to successive invocations and a passive flow where control is passed back to previous invocations and a solution is built, so students' definitions were analysed for the following points:

- A description of the active flow of control.

- A mention of changing parameter values or breaking the problem into smaller instances of the same problem.
- A mention of the base case.
- A description of the passive flow of control and how the solution is built up in reverse.

Very few definitions contained references to all these aspects of recursion, and although many were unclear, this did not indicate that students could not trace recursive programs accurately. Thus students' written definitions of recursion were not going to give an indication of their mental model.

Tracing methods

Students used different methods to trace the different recursive programs. This obvious feature of each trace was recorded as the tracing method might prove to be an indication of a student's mental model. Five different styles of tracing were identified (and an "other" category for idiosyncratic trace methods) but later analysis established that the tracing method a student uses has little bearing on their mental model. Thus the analysis of the tracing methods identified from the 2001 test data are included as an appendix (see Appendix B).

Descriptions of programs

Students' descriptions of the programs were mostly wrong and some were obviously guessed. Some students used their trace to describe the *specific* program call's behaviour instead of giving a description of the program's general behaviour. For example, students said `program3(n)` returns `n-1` although they had evidence from their own traces that this was not the case. Most students could not describe the general behaviour from a specific instance of the program but analysis of this is left as an area for future research.

The following sections describe other observations that were made when analysing this data.

Difficulty dealing with complexity of nested function calls in Scheme

Students' traces of the following lines of code made it apparent that they had difficulty in dealing with the complexity of Scheme nested function calls:

Program 2: `(if (null?(helper(cdr(helper lst n))n)).`

Program 4: `(cons(cons(car l)(prog4(cdr l)))(cddr l)))))`

Students made many mistakes in the order in which these lines of code should be executed. What made interpreting this code difficult was its layout and some students had bracketed sections of the code in order to determine the correct order of execution. Nine students omitted the second call to `helper` in program 2. A reason for this might be that they thought that the second (linear) call and the second recursive call are the same call.

The analysis of the traces of program 2 showed that this program was not a suitable program for this study. This was because only the function `helper` is recursive and it was fairly simple. Most students traced the execution of `helper` correctly and made mistakes in the line of code shown above. Thus the high number of incorrect answers to this question (see table 4.1) is misleading as these were not necessarily the result of non-viable mental models of recursion but rather the result of Scheme misconceptions. Also a student's answer for a recursive call does not necessarily indicate whether their model of recursion is viable or not.

Misconceptions about parameter passing and return values

There were misconceptions about how solutions to recursive programs are built that were owing to misconceptions about parameter passing and how functions return values. Understanding of parameter passing is a prerequisite to building a viable mental model of recursion. Thus students without viable models of parameter passing, are not able to construct viable models of recursion. Scheme has the advantage of hiding many of the details of the physical machine and should thus be an ideal language for instruction. However, students still need viable understanding of mechanisms such as parameter passing before more complex concepts such as recursion can be introduced.

A standard template of recursion

It seems that some students constructed a *standard* template of recursion where certain aspects of recursion always follow the same rule. These aspects include: how new argument values are calculated, what constitutes the base case and what value was returned at the base case. Examples of templates found are:

- Numerical arguments are always decremented by 1.

- The base case is reached when a numerical argument is 1 or 0 or a list is empty (these students ignore the actual testing condition in the program such as `n <= 2` or `(null? cdr lst)`).

Bhuiyan et al. [1994] found that students often use a *recursion template* to derive recursive programs. This is a schema based on structural analogies to programs they have encountered previously with empty slots for the base case and recursive case. The student then attempts to fill these slots to solve the particular problem at hand. It is interesting to note that students seem to have such a template even for tracing recursive programs where the changing parameter in the recursive call and the base case check are explicit in the program code.

Numbers vs. lists

Students had more difficulty tracing programs that manipulated lists than numbers. Table 4.1 shows that programs 1 and 3, which manipulated numbers, had a greater percentage of correct solutions than programs 2 and 4 which manipulated lists. There are a number of possible reasons for this[1].

- Numbers are easier to deal with as at every stage of the recursion there is one number passed as an argument and during the passive flow, one number is passed back to each previous invocation. Lists are more difficult as entire lists are passed as arguments and passed back as return values.

- The programs that manipulated numbers were simpler (they did not include extra function calls such as `cdr, car, cons, null?`). It could be that the added complexity that these function calls introduced was the cause of mistakes.

Two recursive calls

Some students had the misconception that when there were two recursive calls, such as in program 3, there was still only one new copy of the program invoked. The two new arguments (`n-1=3` and `n-1=2`) were passed to this **one** new invocation of the program; the first argument to the first call and the second to the second. So instead of each instantiation creating two new instantiations, there were two parallel lines of calls. Investigating this further is left as a suggestion for future research.

[1]Program 2 had a high number of incorrect traces because students did not call `helper` the second time. This error had nothing to do with the fact that the data manipulated was a list.

4.3.2 Interviews

The issues brought to light from this analysis provided the input for developing questions that would be asked during interviews. I wanted to determine whether students recognised how head-block and tail-block commands influenced the way in which recursive programs execute and thus whether they had an understanding of the two flows of control. I thought that observing a student trace a recursive program and asking questions about their mental processes at every step of execution would reveal more information about their mental model. Interviews have been used in previous mental model research to gain a greater understanding of students' thought processes [Kahney, 1989; Bhuiyan et al., 1991; Dicheva and Close, 1996].

Finding volunteers was problematic. Students who were willing to take part in a private interview were noted and contacted. Only five students volunteered and out of these only two could be contacted. One had de-registered from the university. The other was interviewed, but he had a very good understanding of recursion and therefore provided no insight into understanding how non-viable mental models were constructed. The class was asked again for volunteers and one other student came forward. This student again had a very good grasp of recursion and therefore also provided no insight into the construction of non-viable mental models. I concluded that interviews would not provide suitable data, as students who volunteered were those with viable models of recursion. Another shortcoming of interviews was that students did not express their understanding clearly making it difficult to gain insight into their knowledge.

4.3.3 Conclusion and suggestions for next stage

At this stage some traces seemed to indicate a copies model or looping model, but other models and a method to identify them were not yet clear. I had been analysing students' definitions of recursion, which only would have indicated a student's mental models if students were been more articulate in expressing their thoughts. I had also hoped that interviews would provide me with an in-depth understanding of student thinking but as volunteers were not forthcoming, another source of data was necessary. Exam scripts were an obvious source as the FAC exam always included a recursive trace question. Permission to obtain the June 2001 FAC exams scripts was obtained and these provided a large amount of data since all registered students had written the exam. Also, the recursive trace question was one of the easier ones in such an exam, and so students tended to complete it fully.

```
Algorithm1(numlist)
      if numlist is empty then
         1
      else 2* head(numlist) || Algorithm1(tail(numlist))

Trace this program with Algorithm1( (4 1 3 5) )

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

Figure 4.1: Q2– List manipulation recursive program – June 2001 exam

One drawback of this data was that it would not allow study of how prior programming experience influences mental models as each script was anonymous. Anonymity was a requirement of the University Ethics Committee to obtain this confidential data. Also, since the exam contained only one recursive trace question, the question of whether students constructed more than one model and used them depending on the question, could also not be studied. There was also not a question asking students to define recursion.

4.4 Stage 2

One hundred and seventy two traces were obtained from the 2001 FAC exam. The recursive question was the list processing algorithm in Figure 4.1.

4.4.1 Development of coding method

From the informal analysis in stage one, and initial analysis of this data, a coding method was developed. Categories were defined and each exam trace was coded into the categories it featured. The categories were derived from the definition of recursion, using the same points that students' definitions of recursion were analysed for in section 4.3.1 above. As coding progressed, new categories were defined from features found in traces and all data was recoded. Table 4.2 lists the final categories.

4.4.2 Validation of categories

At this stage I asked three experts (the course lecturer, another lecturer from the School of Computer Science and a course tutor) to validate coding categories I had developed

Active flow	
Copy	a new invocation with a new argument shown
Loop	operation is done on the list element by element
Not shown	only answer given or the trace is not detailed
None	no recursion, one or two step evaluation
Null	nothing can be concluded
Algebraic	algebraic manipulation of function call
Base case	
Stop	recursion stops once base case is reached
Switch	once base case reached, switch from active flow to passive flow
Check incorrect	incorrect test for base case
Base omitted	operation at base case omitted
Passive flow	
Copy	a partial solution is calculated at each level and returned to the previous invocation
None	solution evaluated at base case
Return values	each invocation's return value saved and used in calculating a solution
Return problem	misconceptions about parameter passing and return value evaluation
Operation changed	$\|\|$ changed to $+$ or $\times$ or combination, or order of operations changed

Table 4.2: Coding Categories

and to verify initial coding. The course lecturer found evidence for a model which he had found in the past. He termed a "two step" model where students execute a recursive program in two steps. They evaluate the recursive call without re-invoking the recursive program and incorporate the base case somehow. He had observed that students often perform only these two operations without any active or passive flow. Exam scripts were re-examined and evidence of this was found. New categories to describe this were added and scripts were re-coded.

The other lecturer suggested that each trace could be one of the following two types:

true **trace** : students show some or all aspects of program execution

evaluation of solution: students show how an answer is calculated without giving much indication of how the computer builds this solution.

These two types provided additional insight and facilitated the identification of mental models. Mental models could be more reliably identified from "traces" because these showed the program invocations and the order in which the solution was

	total	% correct	% incorrect
Answer only	21	67	33
Trace	84	79	21
Evaluation	16	38	62
Trace/Evaluation	51	96	4

Table 4.3: Traces and evaluations in 2001 exam

built. This clearly indicated the student's understanding of the active and passive flows of control. Students evaluated a solution if they could not show the mechanics of recursion or they did not understand what was required of them when asked to "trace the program's execution showing all workings". Exam scripts, where the student had only given an answer and showed no indication of how this answer was obtained, were excluded.

Many students traced only the active flow and, once the base case was reached, simply evaluated the answer, showing no passive flow. It was difficult to determine whether this was because their model did not include an understanding of the passive flow or because they could see that the correct answer is available once the base case is reached. A script of the type *trace/evaluation* later became an indicator of the "active" model (see section 4.5.2)

Table 4.3 indicates which exams were disregarded as they contained only a solution and which had been traced, evaluated or had a combination of both (mostly a trace of the active flow and then an evaluation of the solution without an indication of the passive flow). Most students did trace the program execution and they reached the correct solution. Of the students who evaluated an answer, many made some mistake and thus could not reach the correct solution. Nearly all students who traced the active flow and then evaluated an answer obtained the correct solution. Thus it would seem that a mental model that does not account for the passive flow was viable for this program.

4.4.3 Mistakes

It became apparent that some of the coding categories indicated mistakes made by students who nevertheless did have a viable model. Examples include omitting the base case or checking an incorrect condition for the base case. Mistakes made with operands such as `head` and `tail` are the result of misconceptions about Scheme functions and not recursion. Another frequent mistake that was made was to change the concatenation operation to addition or multiplication. The trace in Figure 4.2 shows

```
Algorithm1( 4 1 3 5)
( 2 * 4)+ (Algorithm1( 1 3 5))
( 2 * 4)+ (2 * 1) +(Algorithm1( 3 5))
( 2 * 4)+ (2 * 1) + (2*3 )+ Algorithm1( 5)
( 2 * 4)+ (2 * 1) + (2*3 )+( 2*5) + Algorithm1()
( 2 * 4)+ (2 * 1) + (2*3 )+( 2*5) + 1
=( 2 * 4)+ (2 * 1) + (2*3 )+11
=( 2 * 4)+ (2 * 1) + 17
=( 2 * 4)+ 19
=27
```

Figure 4.2: A trace with a mistake but a viable model

such a case. This trace shows that the student has understood the correct flow of control (both active and passive) and thus possesses a copies model, only the operation performed on the list elements is incorrect. Mistakes, such as these, do not necessarily indicate non-viable mental models.

On the other hand, coding categories such as “Return value calculated at each level” and “Problem with parameters/return values” indicate misconceptions that prevent students from constructing viable models of recursion.

4.4.4 Conclusion and suggestions for next stage

After the process of coding the exam data into the categories it was possible to develop a method of identifying mental models. Figure 4.3 provides examples of traces from which copies and looping models can be identified and Figure 4.4 where an active model can be identified.

Scripts coded into the *copy/switch/copy* categories indicated the copies model. These traces showed evidence of understanding of the active flow, with recursive invocations of the program with new arguments, the base case was correctly identified and processed, and the solution was built as control was passed back during the passive flow.

The looping model was indicated by the combination of *looping*, *stop* and *none* categories. The recursive operation was executed on each list element in turn, the recursion (actually the iteration) stopped at the base case and there was no passive flow.

Students whose exams were characterised as *trace/evaluation* and coded as *copy*,

```
Algorithm1( 4 1 3 5)
(2x4) || Algorithm1(1 3 5)
|  |  (2x1) || Algorithm1(3 5)
|  |  |  (2x3) || Algorithm1(5)
|  |  |  |  (2x5) || 1
|  |  |  |  |
|  |  |  |  ->(10 1)
|  |  |  -->(6 10 1)
|  |  ---->(2 6 10 1)
|----->(8 2 6 10 1)
```

Copies Model

```
Alg ( 4 1 3 5)
=(8 Alg ( 1 3 5))
= ( 8 2  Alg(3 5))
= ( 8 2 6 Alg(5))
= ( 8 2 6 10 Alg( '()))
Answer -> = ( 8 2 6 10 1)
```

Looping Model

Figure 4.3: Copies and looping models from June 2001

```
Algorithm1(4 1 3 5)
 2x4 || Algorithm1(1 3 5)
8 || Algorithm1(1 3 5)
 2x1 || Algorithm1(3 5)
8 || 2 || Algorithm1(3 5)
 2x3 || Algorithm1(5)
8 || 2|| 6 || Algorithm1(5)
  2x5 || Algorithm1()
8 || 2 || 6 || 10 || Algorithm1()
  if numlist empty
     then 1
8 || 2 || 6 || 10 || 1
    therefore ans (8 2 6 10 1)
```

Figure 4.4: Active Model from June 2001

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

 Trace this program with n=8
```

Figure 4.5: Q1 – Recurrence relation recursive program – June 2000 exam

stop and *none*, indicated a new model which was named the "active" model. Students with this model correctly indicated the active flow but did not show the passive flow. They calculated a solution once the base case was reached.

Other combinations of codes suggested other models but before defining them precisely, I wanted to analyse more data.

An issue considered to be significant was that many students had not shown the passive flow but had nevertheless correctly computed the solution. Thus, for this program, a mental model of recursion that didn't account for the passive flow seemed to be viable. One reason for this might be that this program had the recursive call at the end of the program with no further commands in the tail-block. This program described an action (multiply by two) on every list element and once the student realised this they could easily evaluate the answer. If the student interpreted the question as *"what does this function call return?"*, then they would simply evaluate an answer. In the case of embedded recursion, with the recursive call in the middle of the program and a command in both the head- and tail-blocks, the base case must be reached and each recursive invocation's return value calculated and passed back to the previous invocation before the solution can be determined. Thus the order in which the solution is built during the passive flow is important and it would seem that students are more likely to show the passive flow when tracing this type of program. In the next stage, traces of such an embedded recursive program were analysed.

4.5 Stage 3

There were 162 exam scripts from the June 2000 FAC exam. The recursive question students were asked to trace was a mathematical recurrence relation with embedded recursion, see Figure 4.5.

```
if n= 1  then 1
 4*c(n/2) + 3 = 1
 4 c(1/2) + 3 = 1
        c/2 = -1/2
          c = -1

 Then if n = 8
 4(-1)(8/2) + 3 = -13
```

Figure 4.6: An algebraic manipulation of the recurrence relation

4.5.1 Coding the data

Coding began with the coding categories previously defined, as well as the *answer*, *trace*, *evaluation*, and *trace & evaluation* characterisation. As analysis proceeded a new category was added, the "algebraic" category, and the "operation changed" category was interpreted differently.

4.5.1.1 Algebraic category

Many students interpreted the recursive function call as an algebraic expression and obtained a solution from mathematical manipulation. Instead of calling the function with the argument `n/2`, these students calculated the expression: `4*c*n/2+3 = 16c+3` or simply dropped the `c` and calculated `16+3 = 19`. This characteristic of a trace was an indication of the "step" model or the "return value" model which are described in section 4.5.2 below. The course lecturer suggested that the problem students had with this program was that the function was defined with the letter "c" and students would be more used to functions defined as f, g or h. This would be investigated in Stage 4.

Other students attempted to derive a closed form of the recurrence relation and then derived an answer to the program from that formula. This was called the "algebraic" model and was very rare. For example one student derived the formula $2^8/2 = 2^7$ since it gives the correct result of 127.

Another example of an algebraic manipulation is shown in Figure 4.6. Students who manipulated the program in this way had not assimilated new knowledge of recursion into their existing cognitive framework and were simply applying their knowledge of algebra.

4.5.1.2 Changes operation – the magic model

Students did not change the adding and multiplying operations of this question (as students the previous year had changed the concatenate) but they changed the order in which these were executed as the recursion exited during the passive flow. The correct operations in order gave 127 as a solution:

```
4*(4*(4(1)+ 3)+3)+3 = 127
```

Some traces showed some understanding of how the recursion executes and how a solution should be calculated, but with some important aspects missing. These aspects are *magic'ed* away. These traces resembled those of students with the looping model. The difference was that instead of an answer being reached at the base or the answer being the base case, there was some indication that the solution was built up from the base case. Often it was how this solution was built that the student did not fully understand, that is, the passive flow. Traces such as these indicated a *syntactic magic* model. These students recognised the recursion and had some ideas of what this entailed, most obviously a number of instantiations with changing argument values, and then some sort of calculation of a solution, but without a clear idea of how the program achieves its effect [Kahney, 1989]. The following types of magic model were found. They indicated progressively more "magic" models of recursion.

- `4*4*4*(1)+3+3+3 = 73`

  Students with this magic model were very close to constructing the copies model. The order of the calculation of the solution was incorrect because of lack of understanding of the passive flow.

- `4*4*2*1+3 = 35` or `4*(4*(2)*4*(1)) + 3=131` or `4*4*4(1+3)=256`

  Students with this magic model failed to take the entire function call into account when building the solution. Also, there was a misconception about parameters as the parameter values were taken to be the return values from the successive function calls.

- `3+3+3+1 = 10` or `4(1)+3+3+3 = 13`

  Students with this magic model understood that there was an active flow and usually had the correct number of invocations, but could not show how the parameters were involved in this process. There was also a lack of understanding of how the statements around the recursive function call (the head and tail blocks) were involved in the calculation of the solution.

4.5.1.3 Traces versus evaluations

Again, classifying each script as a trace or evaluation or combination of the two aided the identification of mental models. To achieve the correct result for this program there **had** to be a full trace of the program execution, as opposed to the June 2001 question were an evaluation could result in a correct answer. Thus any script classified as an *evaluation* or *trace & evaluation* indicated a non-viable model. A viable model did not necessarily mean the answer was correct, as mistakes, that were not misconceptions about recursion, could still have been made.

4.5.2 Identification of mental models

The combination of categories that each trace had been coded into, made it possible to identify the students' mental models of recursion.

The copies model

This is the model that Kahney hypothesised and showed that experts possessed. Students with this model had constructed a viable understanding of the active flow, the base case and the passive flow of recursion.

Looping model

This is the model that Kahney hypothesised and showed that novices construct. Students with a looping model view a recursive procedure as a single object rather than a series of instantiations and thus see recursion as a form of iteration. Students with this model obtained the solution once the base case was reached, thus see the base case as the stopping condition of the loop. There was often no indication of repeated invocations of the program or diminishing arguments.

Active model

Many students did show an understanding of the active flow of control of recursion and the instantiations of the recursive functions with smaller argument values until the base case is reached, but did not understand (or did not show) how the solution was built during the passive flow. These students could have a copies model, but had determined the solution without giving a trace of the passive flow.

Return value model

This model stems from misconceptions about what happens to return values from a function call. Many students hold the misconception that at every function call some value is calculated and stored and these values are then combined (for example, by adding or multiplying) into a solution once the base case has been reached. Students who possess this model often changed the operation in Q2 from concatenation to addition or multiply. Students who possess this model cannot construct viable models of recursion until they construct a viable model of parameter passing and the control stack. Although Scheme does hide much of this detail from the programmer, an understanding of parameter passing and function instantiation and return values is a prerequisite for the construction of the copies model of recursion.

Syntactic magic model

Students with this model, also defined by Kahney, recognise programs with recursive syntax but trace them without a clear idea of how the program achieves its effect. Thus they do not really have a model for the behaviour of a recursive program but recognise syntactic segments as indications of recursive behaviour and use their "magic" ideas of recursion to determine a solution. This model is discussed further in 4.6.1

Step model

Students with this model have no concept of any recursive flow of control. They simply evaluate the IF-THEN-ELSE and execute either the recursive condition once (one-step), both the recursive condition and the base case (two-step). This model discussed further in 4.6.2

Algebraic model

Students with this model either attempted to derive a formula that allowed them to calculate the result of the program call or manipulated the algorithm as an algebraic equation. This model was extremely rare and only found with programs that involved calculations and not in ones that manipulated lists. What was more common was that students manipulated the recursive function call as if it were an algebraic function (see section 4.5.1.1 above). This did not indicate the algebraic model but was a characteristic of traces displaying a step model or a return value model.

Copies	Looping	Active	Return	Magic	Step	Algebraic	Odd
51%	10%	0%	6%	9%	17%	3%	4%
June 2000 (n=162)							
Copies	Looping	Active	Return	Magic	Step	Algebraic	Odd
44%	13%	33%	2%	3%	1%	0 %	4%
June 2001 (n=151)							

Table 4.4: Mental models identified

Odd models

These are also models described by Kahney [1989] and are held by students with idiosyncratic ideas about some features of the programs and thus do not correctly predict the behaviour of the programs. Traces coded as identifying odd models often showed aspects of looping, algebraic and return value models, or were simply incomprehensible.

Table 4.4 provides a summary of the models that were found from the June 2000 and 2001 exams. Because the two exams had different recursive questions, some models were more prevalent in one data set than the other, but there was evidence of most models in both.

The results from 2000 and 2001 cannot be directly compared as they are drawn from a different group of students and from two very different recursive programs but the following observations can be made:

- Over the two years teaching has resulted in the majority of students constructing a viable copies model. For the June 2001 question an active model and a looping model could result in the calculation of the correct answer, so these models were also viable for this particular question and thus 90% of students in 2001 had viable models.

- The high number of students with a step model (17%) in 2000 is a cause for concern. A step model indicates that a student has not been an active participant in learning and has therefore constructed no viable knowledge about a concept fundamental to their course of study.

- The 9% who constructed magic models in 2000 have some understanding of recursion and, with instruction directed at their misconceptions, could construct a viable model of recursion.

- Those students with return value models did not have a model of the behaviour

of the computer as it executes recursive functions. A viable model of computer behaviour is a prerequisite to the construction of a viable model of recursion

- The looping model is the model that Kahney [1989] hypothesised that novices have. Thus it is not unexpected that some students possess this model. This model could be problematic as it can be viable for certain programs but not for others. This is discussed further in Chapter 5

4.5.3 Conclusion and suggestions for next stage

After this stage, the following questions needed to be addressed:

- Had the 2002 first years constructed the same mental models of recursion as the 2001 and 2000?
- Do students construct different mental models and use them selectively depending on the recursive program they are asked to trace?
- If the function were defined as `f(n)` instead of `c(n)` would students make fewer mistakes?

The decision was made to give the 2002 first years both Q1 and Q2 in one of the regular first year class tests.

4.6 Stage 4

The recurrence relation from the June 2000 exam (Q1) and the list manipulation question from the June 2001 exam (Q2) were given to the 2002 first years in a class test. Half of the papers had the function for the recurrence relation program defined with an “f” and the other half had it defined with a “c” in order to investigate the effect this had. A class test was used to ensure the maximum participation and 169 students wrote the test. All students had completed Q1, but 20 had left out Q2 due to lack of time or choice, and these were therefore disregarded from the analysis of Q2.

This data was coded directly into the mental models that had been identified: Copies, Looping, Active, Return Value, Magic, Step, Algebraic, Odd. This class was not as familiar with recursion as the previous two as this class test was given at the beginning of May instead of in the June exam (by June they would have had more practice with recursion and would have studied for their exam). Thus more non-viable models and more mistakes were expected. The same mistakes and misconceptions

```
2& head(4 1 3 5 ) || alg1 ( tail(numlist))
2*4 || 2 * 5
  8 || 10
 -> 2*1 || 2 * 3
   2 || 6
output: ( 8 10 2 6)
```

Figure 4.7: Trace of the list manipulation with a magic model

were observed. Analysis of these traces made it possible to define the magic and step models, discussed below, more precisely. Some of this stage's analysis resulted in re-coding of the June 2000 and 2001 data, thus the values in table 4.4 are those calculated after Stage 4.

4.6.1 The magic model

The syntactic magic (or just magic) model categorised students who recognised programs with recursive syntax but whose trace indicated they had no clear idea of how the program achieves its effect. There was always an indication of an active flow, often with some misconceptions about parameter passing or how successive argument values are calculated. The base case was usually seen as an indication that the active flow must stop. There was little understanding of the passive flow and how return values build a solution. The types of magic models found in the recurrence relation question were listed in section 4.5.1.2.

Students with magic models for Q2 also often had misconceptions about `head`, `tail`, parameter passing and return values. But, none of these traces incorporated the base case value in any way. Figure 4.7 provides an example.

4.6.2 The step model

In Q1 those students with a step model simply executed the function linearly in either one or two steps. Those with a one-step model determined that the ELSE part of the IF-THEN-ELSE construct should be evaluated since `n=8` and `8` is not equal to `1`. The solution was given as `4*c(4) + 3 = 16c+ 3` or as `16 + 3 = 19`. Those with a two-step model also use the base case return value in calculating their solution as shown in Figure 4.8 A. Figure 4.8 B provides an example of a student with a step model's trace of Q1.

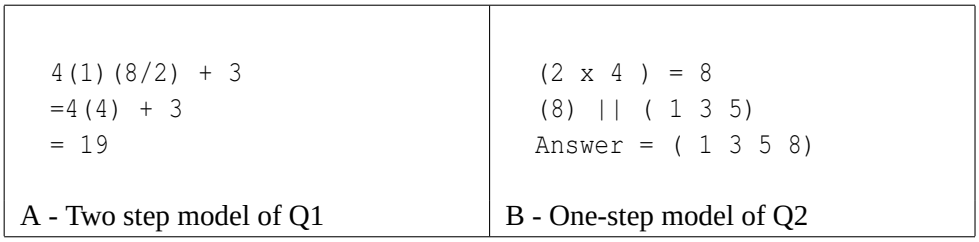

Figure 4.8: Tracing with a step model

	Copies	Looping	Active	Return	Magic	Step	Algebraic	Odd
	46%	0%	4.5%	15%	5%	26%	0.5%	3%
f(n)	26%	0%	2.3%	7.5%	1.5%	12.4%	0.0%	1%
c(n)	20%	0%	2.2%	7.5%	3.5%	13.6%	0.5%	2%
Q1 - recurrence relation (n=169)								
	Copies	Looping	Active	Return	Magic	Step	Algebraic	Odd
	26%	14%	25%	5%	8%	11%	0%	11%
Q2 - list manipulation (n=149)								

Table 4.5: Mental models in 2002

4.6.3 Mental models in 2002

Table 4.5 summarises the models identified from the 2002 test data. For Q1, the copies model was the only viable model. For this question just under half the students have a viable model. For Q2, the loop and active models could also be viable in that they allow the student to determine the correct answer. Therefore 65% have viable models (although not all determined the correct solution). Unfortunately, there was a high percentage of students with non-viable models. One reason for this could be that this test was written in May and thus students had not had enough time to learn this concept and had thus not yet constructed viable knowledge. This could account for those students who had magic models. Students with return value models had not yet constructed viable models of computer behaviour, something that is a prerequisite to understanding recursion. Students with step models had not constructed any viable knowledge of recursion which could either mean they had not been active in their learning or had serious barriers to learning. This type of test could provide a diagnostic that identifies students with misconceptions who could then be provided with special tutorial help aimed at rectifying those misconceptions and thus facilitate the construction of viable models of recursion.

Same model	Copies	31	
	Active	2	
	Step	10	
	Return value	1	49
	Magic	3	
	Odd	2	
Different models	Copies (Q1) + Viable (Q2)	34	
	Non-viable + Copies	4	
	Non-viable + Active	12	
	Non-viable + Looping	17	100
	Copies + Non-viable	6	
	Non-viable + Non-viable	27	
Q2 not completed/traced			20

Table 4.6: Mental models – do students construct more than one?

4.6.4 Function definition and algebraic manipulation

Half of the tests defined the function in Q1 as `c(n)` while the other half as `f(n)`. This was to determine whether the use "`c`" was a source of confusion. Table 4.5 shows that this did not greatly influence the mental models that were identified.

4.6.5 Do students have more than one mental model?

Table 4.6 summarises the mental models identified for each of the fully completed tests. The results indicate that most students do construct more than one model. In some cases two very different models were identified from the two traces. Figure 4.9 shows a student's two traces. A copies model from the trace of Q1 (although a mistake has been made in line five – a `4*` has been omitted), while the trace of Q2 indicates a step model and a misconception about the function `tail`. Figure 4.10 shows two different students' traces of Q1. Both students' traces of the list manipulation clearly showed a copies model, while their traces of this question show magic models. Student A has realised that there should be four instantiations of the program but could not correctly execute the head- and tail-block commands to arrive at the solution. The head- and tail-block commands have been *magiced* away and the base case has been added together four times. Student B's trace shows some understanding of the active flow but there are misconceptions about parameter passing. The tail-block command (`+3`) has been ignored during the passive flow. Students might construct different models for different types of recursive programs depending on how much practice they have had with each type and what misconceptions they have.

```
n=8
4 x f(n/2) +3
4 x f(4) +3
4 x (4 x f(2) + 3) + 3
4 x (4 x (f(1) + 3) + 3) + 3
4 x (4 x (1 + 3) + 3) + 3
4 x (4 x (4) + 3) + 3
4 x 19 + 3
```

Q1 - Copies model

```
Numlist is not empty
2 * (head(numlist)) || Algorithm1(tail(numlist))
2 * 4 || 2 * 5
8 || 10
output list (8 10)
```

Q2 - Step model

Figure 4.9: One student's traces of Q1 and Q2

Students do not always have only one unique model of recursion. The specific features of the program being traced influence how the student perceives that recursion executes. So they do not see recursion as a process that executes in the same way no matter what data is being manipulated or how this is manipulated. Students construct new models or enhance existing ones as new material is learnt [Bhuiyan et al., 1991]. But new models are not persistent and students will only apply new ones when old ones no longer work. So it is not unlikely that during learning students have and use more than one model.

4.7 Concluding Remarks

Previous research identifying mental models had successfully used students' traces (or interpretations) of specifically developed recursive programs and student-generated recursive programs. Thus a test was developed to collect data for this research. This test contained four very different recursive programs and it was thought that the data gathered from the test would be sufficient for identifying mental models. Additional in-depth understanding was to be obtained through interviewing students as Kahney [1989] and Dicheva and Close [1996] had done. The data gathered from the test was

```
 4 * c (n/2) + 3
= 4 * c(1) + 1 +1 +1) +3
= 4 * 4 + 3
= 19
```

Student A : Magic Model

```
n=8
4 x (8/2) +3
| 4 x 4 + 3   n=4
|  | 4 x 2 + (3)   n=2
|  |   |  4 x 1 + 3
|  |   |    = 4 +3
|  |   -------------
|  |           7 x 4
|   ---------------
|       ( 7 x 4 ) x 4
--------------------
     it will return 7 x 16 =
```

Student B : Magic Model

Figure 4.10: Two examples of traces of Q1

not sufficient because it was administered in a lecture where only 55 students were present, the time given was not sufficient and, because it was voluntary, the students did not answer the questions completely. Interviews did not provide additional information as those few students who did volunteer had viable models.

Although the next data source, exam scripts, did provide rich and plentiful data, I had no control over the questions used and there was only one such question in each exam. Fortunately the 2000 and 2001 examinations had very different recursive questions revealing the many different misconceptions that students could have and allowed for the development of a coding method and from the coded data, a method for identifying mental models of recursion. Unfortunately the two sets of data were not comparable as they came from two different groups of first years.

Thus a third set of data was collected from the 2002 first year class. The same two programs were incorporated into a class test thus ensuring that they would be completed fully. This data allowed the comparison of mental models that students applied to these two different recursive programs.

Throughout the data collection process, data was analysed and a method for identifying mental models was developed in the third stage. As data was collected and analysed the analysis process became more formal and each trace was coded into categories developed from a definition of recursion and from characterises of traces. It was the specific combination of these coding categories that identified the mental model. By the end of this process the following mental models had been identified: Copies, Looping, Active, Return Value, Syntactic Magic, Step, Algebraic, and Odd. The following chapter discusses each of these models in detail.

Chapter 5

Discussion of Mental Models of Recursion

5.1 Introduction

This chapter discusses the mental models that have been identified by this research and examines the misconceptions that are likely causes of non-viable models. Suggestions of how lecturers could use this knowledge to inform their teaching practice and curriculum are made.

5.2 The copies model

To a constructivist educator "shared meaning" is a misleading term as constructivism posits that each student will construct his or her own idiosyncratic meaning. Nevertheless, lecturers do want students to reach a "consensual domain" where there is a similar interpretation of a given concept within the group [Von Glasersfeld, 1992]. Recursion has an active flow where control is passed forward to successive invocations and a passive flow where control is passed back to previous invocations and a solution is built. The copies model is the only consistently viable model of recursion as it models both the active and passive flows of control. Thus students should construct a model that is similar (although it need not be identical) [Ben-Ari, 1998] to the copies model.

While the majority of Wits students in 2000, 2001 and 2002 did construct a copies model, many non-viable models were also constructed. This may be because, when recursion is first taught, simple programs are used as examples. Students construct a model that allows them to obtain correct solutions for those simple programs, but their model could be non-viable for a more complex program. It is at this stage that

educators need to present students with programs or problems that will expose any inadequacies in those non-viable models. Thus lecturers should ascertain the misconceptions that students hold and be aware of the non-viable models they are likely to construct. Lecturers should then ensure that students are shown other recursive programs that challenge any non-viable models.

Previous studies have identified the copies model as the only viable model of recursion [Kahney, 1989; Kurland and Pea, 1989; Bhuiyan et al., 1994; Dicheva and Close, 1996]. This study has also identified the copies model as a viable model, but proposes that the active and looping models may be viable for certain recursive programs. Because the looping and active models are not consistently viable models, they are termed "risky" viable models. These are discussed next.

5.3 The looping and active models

The looping model, defined by Kahney [1989] as the model that novices construct, was also identified in studies by Bhuiyan et al. [1994] and Dicheva and Close [1996]. It stems from the misconception that recursion is a form of iteration. If a student was already familiar with iteration they might construct a looping model, but even students without prior programming knowledge construct this model. While Bhuiyan et al. [1994] divided the loop models into four sub-types (proper, embedded, truncated procedure and two loops) this research identified only the "proper" loop model. Although equating recursion to iteration is a misconception, a looping model can be "viable" if students are asked to calculate the correct output of a recursive program. On the other hand, if students are asked to show the flow of control of the execution of a recursive program, this model will not be viable. Unfortunately, students often believe that reaching a correct solution is more important than how that solution is reached.

The active model could be seen as the "curtailed" copies model as it models the active but not the passive flow of recursion. Although every recursive program has a passive flow, this could simply involve passing control back to a previous invocation and no other calculation. In this case it would not be necessary to trace the passive flow, as the solution would be complete once the base case is reached. Therefore, it could be that students whose traces show only the active flow, do have a copies model and have realised that showing the passive flow is not necessary in these cases. These students would show the passive flow when it is necessary in the calculation of the solution. On the other hand it might indicate that students do not have an understanding of the passive flow at all.

Both these models are “risky” viable models as they are viable for tracing some programs but not for others. If too many simple programs (without commands in the tail block, i.e. commands that have to be executed during the passive flow) were used as examples, students would believe that a looping or active model was viable. If the student is not presented with problems that show the inadequacies of these models, they will continue to use them even when they are not viable. Also, students need to be taught that a “trace” requires them to show the recursive process clearly and not simply give an answer. If students think that they simply have to calculate an answer then they might maintain one of these risky-viable models.

5.4 The magic model

It is impossible for lecturers to **make** students learn. Students are shown many examples in lectures and expected to practice further in tutorials and lab sessions because learning must involve active participation on the part of the students. In order to encourage active participation, the constructivist teacher should choose course material that fosters an interest in learning. Students with a magic model have not been active enough participants in learning. They show an understanding of certain key aspects of recursion, such as the diminishing arguments, the role of the base case and the calculation of a solution, but they do not have a clear idea of how these are applied as a recursive program executes. Most students with the magic model *magic* away what happens during the passive flow. A suggestion for lecturers is to explicitly name the passive flow and make it very clear during demonstrations.

There is a range of magic models with each one *magicing* away more detail of the recursive process. The first, most like the copies model, disregards the passive flow. Students may construct this model if they have not seen enough “live” traces of recursive programs. This might be because they have not attended sufficient lectures or have copied lecture notes that give them a static representation of the dynamic recursive process. A student with this model traces the execution of Q1 as:

```
4*4*4*(1)+3+3+3 = 73
```

The number of instantiations is correct, but the dynamic nature of the calculation, indicating the order in which the solution is calculated during the passive flow, is missing, resulting in an incorrect solution.

The next in the range of magic models shows that the student has misconceptions about how the tail-block of the recursive call is involved in building the solution (as well as disregarding the passive flow):

```
4*4*4(1+3) = 256
```

The head-block and the successive recursive invocations are correct but the tail-block is only executed once and the solution generated without a passive flow. Some students with this magic model also had misconceptions about arguments and added the argument into the calculation:

```
4*4*2*1+3 = 35
4*( 4*(2)*4*(1) )+3 = 131
```

The last magic model found, showed that a student had a misconception about how the head-block is involved in calculating the solution (although the tail-block is taken into consideration):

```
3+3+3+1 = 10
4(1)+3+3+3 = 13
```

5.5 The algebraic model

This model shows the influence of a students' prior knowledge of mathematics on the construction of their mental model of recursion. At Wits, one of the motivations for using a functional programming language as the introductory language, was that it allows for the easy translation of mathematical functions into programs. The benefit of this is that, since students are familiar with mathematics, problem solutions can be defined as mathematical functions that can then be translated directly into program code. This should make learning programming easier. Unfortunately students with algebraic models show a lack of a model of computer behaviour, which could be the result of the strong link demonstrated between mathematics and program code. So, although Scheme would seem to be an ideal teaching language, with its simple syntax and semantics embodying a simple notional machine, the lack of detail about the underlying machine leads to misconceptions. Students with a pure algebraic model are few, but many students used algebraic manipulations inappropriately and incorrectly (interpreting the function calls $c(n)$ as $c * n$) and thus constructed non-viable models.

Algebraic models were identified only from the recurrence relation that manipulated numbers[1]. One type of algebraic model found had students attempting to derive a closed-form formula that would describe the recurrence relation. Students with the

[1]The question that this raises is what model does a student with an algebraic model for a recurrence relation apply to a list manipulation question? Unfortunately the only data that could answer this (2002 class test) had only one student with an algebraic model for Q1 and Q2 had not been attempted - maybe because the algebraic model could not be applied to Q2, but maybe because he or she ran out of time

```
if n= 1  then 1
    4*c(n/2) + 3 = 1
    4*c(1/2) + 3 = 1
    c/2 = -1/2
    c = -1
    Then if n = 8
    4(-1)(8/2) + 3 = -13
```

Figure 5.1: Algebraic model

other type of algebraic model manipulated the entire program as a sort of equation. Students would not have been shown a trace of a recursive program like this in a Computer Science lecture, but they may have seen something similar in a Mathematics lecture and applied it to a recursive trace. This model does lead to seemingly elegant solutions such as the one shown in Figure 5.1.

5.6 The return value model

A viable model of parameter passing and function return values is a prerequisite for the construction of a viable model of recursion. In order to understand the passive flow of control in a recursive program, it is important to understand that each invocation of that recursive program is present somewhere (the student does not necessarily need to understand what the stack is or how it works) and that, at every stage, there is only one value that is being manipulated: the return value from the previous invocation. Students with the return value model have the mistaken impression that, at each invocation, a single answer is calculated and stored. These values are then all incorporated into a solution. This solution is calculated once the base case has been reached.

Students who possess this model cannot trace a program with embedded recursion as no value can be calculated before a subsequent invocation's return value is. In order to overcome this problem they often use algebraic manipulation to obtain a value, thus `4*c(n/2)+3` is interpreted as `4 * n/2 + 3` with the current argument value for `n`. Then, once the base case is reached, all these values are combined (with addition or multiplication) to form a solution. An alternative interpretation of this could be that a mathematics misconception leads the students to calculate a value at each invocation. The presence of these values leads to the misconception that the solution is calculated

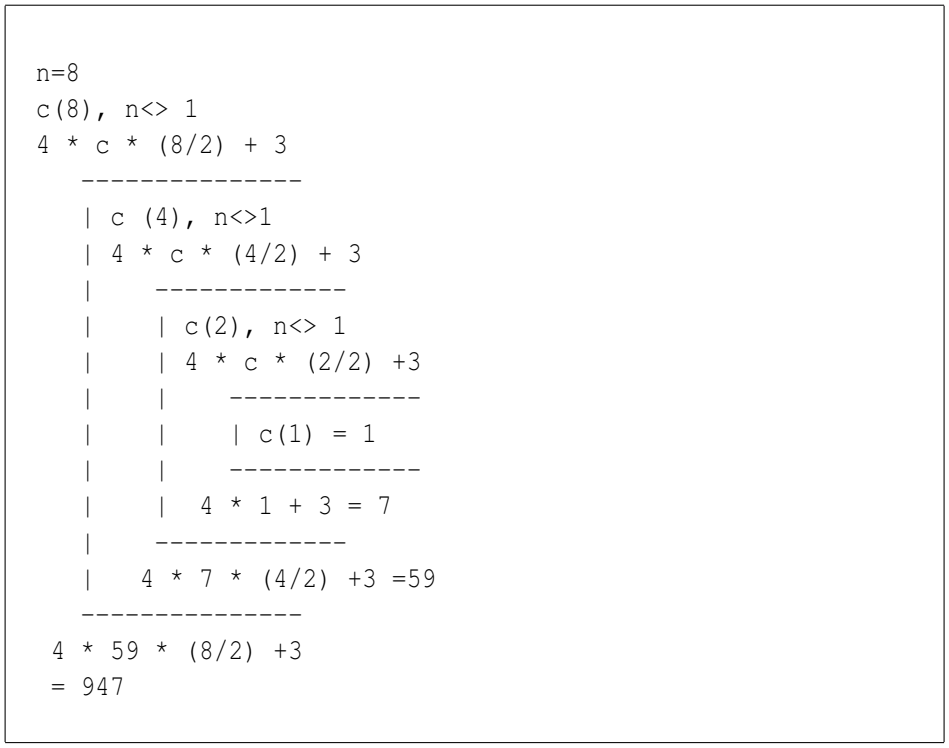

Figure 5.2: A copies model with a parameter problem

by combining them all.

Lecturers should first clearly describe parameter passing and function return values before attempting to teach recursion. Tracing methods, that help to make the idea of the "copies" of a recursive program clear, are also useful as they will clarify how, at each copy, computation is suspended until the subsequent copy has passed back an answer. Again, it is important that students actively participate in lectures, tutorials and lab sessions, as a dynamic view of this process is necessary. Recursion animation or program visualisation tools would also be helpful to illustrate the flow of control dynamically.

Nevertheless, a student can have a misconception about parameter passing and return values but have a copies model of recursion, as the figure 5.2 shows.

5.7 The step model

Students with the magic model and return value model have non-viable models of recursion but with some careful teaching and emphasis on the passive flow these students' non-viable models could be challenged encouraging them to construct a viable

model. The step model, in contrast, is constructed by students without an understanding of the repetitive nature of recursion and thus would need much more instruction and active participation before a viable model is constructed.

Students with the step model show no understanding of the active flow, the base case, or the passive flow of recursion. Those with a one-step model evaluate the IF-THEN-ELSE construct, realise that the IF-part does not apply and simply evaluate the THEN-part. The recursive call in the THEN-part of the programs is ignored and a solution is reached in one step. In the case of numerical calculations in Q1, students often had an algebraic misconception and calculated an answer by changing the function call into an algebraic variable or ignoring it altogether. For the list processing program they did the operation on the first element of the list (and sometimes also on the last), ignored the recursive call and simply concatenated with the remaining list. It is hard to believe that these students had attended many lectures, tutorials or lab sessions as demonstrations of the execution of recursive programs would have made the active flow very clear.

This model is similar to the *simple* model found by Dicheva and Close [1996]. They found that children with the *simple* model did not iterate at all and simply evaluated the inputs linearly. The recursive call is ignored but the calculation on the argument to that call is executed.

The *do-it-once-more* defined by Dicheva and Close [1996] was not found. Children with this model have the misconception that a recursive call means "execute the command in the head-block only once more" because children overgeneralise a principle from another related area of LOGO programming (i.e. their understanding of procedures).

5.8 Odd models

Students with odd models often hold many misconceptions that, when added together, confuse them so much that their trace is highly idiosyncratic. For example:

```
2 * head(numlist) || Algorithm1( tail(numlist))
2 * 4 ( 4 1 3 5 ) || Algorithm1 ( 5 ( 4 1 3 5))
The output is: 32 8 24 4 20 5 15 25
```

This student took the list and multiplied it by $8 (2 * 4)$ and made a mistake of not multiplying the last element by 8 and then the entire list by 5. Again, it is hard to

believe that this student has seen many examples or practiced him or herself. It is more likely that his or her model was constructed without any facilitation from the lecturer or tutor.

5.9 Evaluating solutions

A first year Computer Science student needs to construct a viable mental model of recursion in order to use recursion as a problem solving technique and to implement recursive programs. It would seem that if a student can determine the solution of a given recursive program call that they have constructed a mental model of recursion that is viable, but this is not necessarily the case. It is important that students are taught to trace execution showing both the active and passive flow of control of a recursive program. Such a trace indicates their understanding of how the computer arrives at the solution and in this way teaches a student about how a computer operates. With simple programs, evaluating results may be viable but as recursive programs become more complex, tracing is necessary to reach the correct solution.

5.10 Suggestions for classroom practice

The aim of this research was to identify Wits first year Computer Science students' mental models of recursion, as revealed by their trace of a recursive program. With this knowledge, lecturers gain a greater understanding of learning and how it can go wrong and can use this understanding to enhance their teaching. Just as a learner is an active participant in learning, a teacher must be active and constantly evaluating his or her own practice in order to teach effectively. A teacher who teaches unvaryingly to each and every student will not mediate learning as well as one who is continually gauging his or her learners' understanding, gaining insights into the sources of students' misconceptions and feeding that knowledge back into his or her teaching practice.

A constructivist view of knowledge can make lecturers aware of what their role should be to foster successful learning. Students' existing knowledge and idiosyncratic conceptions need to be identified as they can serve as a basis for further knowledge construction. A non-viable mental model is constructed due to a misconception that a student holds. An educator who is able to identify a student's mental model and who is equipped with the knowledge of what misconception underlies that model could provide students with examples and exercises that clear up the misconception and thus

facilitate the construction of a viable model. As Von Glasersfeld [1992] points out, it is the constructivist teacher's role to generate some perturbation about conceptual structures that are currently being used to attempt to foster new combinations of concepts.

Before teaching recursion, all concepts that are prerequisites for understanding recursion need to be taught. These include function calling and parameter passing. The next step should be to cover embedded function calling thoroughly. Introducing recursion to novices can be done by a simple dramatisation [Ben-Ari and Reich, 1996] and can benefit from visualisation [Velazquez-Iturbide, 2000; Dann et al., 2001] or program animation to make the two flows of control explicit [Wilcocks and Sanders, 1994; George, 2000]. Aspects of recursion such as the base case should be emphasised [Haberman and Averbuch, 2002]. Diagnostic testing can be used to determine students' mental models and then students could be taught in such a way as to specifically to challenge their misconceptions [Bhuiyan et al., 1994; Dicheva and Close, 1997].

As Bhuiyan et al. [1991] found, students construct a particular mental model that is then enhanced as new material is learnt and when models become inappropriate for certain problem solving situations, new models evolve. But since mental models, once formed, are persistent, students frequently return to more established mental models when a first attempt to solve a problem with a new model fails. Thus the range of recursive problems that students are presented with must be well thought-out and cover many different examples to foster the construction of a viable mental model of recursion.

Chapter 6

Conclusion

6.1 Introduction

This research has identified the mental models of recursion that first year Computer Science students at the University of the Witwatersrand have constructed in 2000, 2001 and 2002. This chapter provides a summary of this research – a qualitative analysis of students' traces of recursive programs with the aim of generating knowledge of student learning of recursion and hence informing curriculum and pedagogy. The limitations of this research are discussed and suggestions for extending research in order to gain more from this type of study are made.

6.2 Overview of the basis and motivation for the research

Considering a learning theory such as constructivism can give lecturers insights into student learning. While traditional epistemology would view knowledge as a true reflection of an ontological reality, constructivism holds that the only world we know is that of our experience. Thus, to a constructivist teacher, students are actively involved in the learning process and will construct knowledge that is "viable" rather than "true". Viable knowledge allows students to accurately and consistently model the world of their experience and thus could be idiosyncratic. Communication through language allows a group to construct shared common knowledge.

A student's mental model of a concept or process is their cognitive representation of that concept or process. Recursion is a fundamental concept in the discipline of Computer Science. Students find recursion difficult to learn [Sooriamurthi, 2001; Haberman and Averbuch, 2002; George, 2000]. Studying the mental models of recursion that students construct, and attempting to understand why students' learning can

lead to non-viable mental models is of interest to teachers. The study of mental models can provide teachers with insights into student learning and thus inform curriculum and pedagogy.

This research has developed a method to identify a student's mental model of recursion from their trace of the execution of a recursive program. This is done by coding traces into categories that were developed from a definition of recursion and from aspects of the traces that were analysed. The specific combination of categories in which a trace was coded, was used to identify the students' mental model.

The following section summaries the answers to the particular questions that were posed by the research.

6.3 Summary of findings

The aim of the research was to identify the mental models of recursion which Wits first year Computer Science students constructed, identify the misconceptions that students non-viable constructions were based on, and use this knowledge to inform curriculum and pedagogy.

The research explored the following particular questions to better understand student learning.

Can a student's definition of recursion provide insight into their mental model?

There was a wide variety of definitions of recursion. This could be due to the open nature of the question. What definitions had in common was that they were neither clear nor comprehensive. This indicates that students had poor written communication skills. Some students were not first language English speakers and thus would be further hampered by difficulties in expressing themselves in English.

Unclear and badly worded definitions were expected as Carbone and Hurst [1999] had found that students did not possess the vocabulary necessary to describe their learning and made it difficult for the researchers to explore student learning. Similarly, in this research, it meant that students' definitions of recursion could not be used for the purpose of identifying students' mental models. But, there did not seem to be a link between a student's written communication skills and their understanding of recursion. Many students who gave incorrect or incomplete definitions were nevertheless able to correctly trace the recursive programs.

What viable mental models of recursion can be identified?

Previous research identifying mental models of recursion [Kahney, 1989; Kurland and Pea, 1989; Dicheva and Close, 1996; Bhuiyan et al., 1991] had identified the copies model as the only viable model. This research corroborates their findings but suggests that the active and looping models can be viable in some instances. Thus they have been termed "risky" models as a student with these models could have the false impression that his or her model is viable. A student will have this false impression if too many recursive programs, that can be successfully traced with these models, are encountered.

What non-viable mental models of recursion can be identified?

The non-viable models were the *looping, step, magic, algebraic,* and *return value* models.

Do Wits students construct the mental models identified in previous research?

The looping model was previously identified by Kahney [1989]; Dicheva and Close [1996] and Bhuiyan et al. [1991]. Kahney [1989] also found the magic model, similar to the *syntactic* model defined by Bhuiyan et al. [1991]. The step model is similar to the *simple* model identified by Dicheva and Close [1996]. The algebraic and return value models are models have not been identified in any other studies. This could be a result of the particular recursive programs that were used by this research. Thus the same programs should be given to students from another university to determine whether these are common models. Dicheva and Close studied children programming in LOGO while this research studied students programming in Scheme. Some models were identified in both groups, but since misconceptions stem from the particular programming language used and are brought to light by the specific program students are asked to trace, certain models will only be identified in one group and not another. Thus Dicheva and Close's *do-it-once-more* model was not found, as this stemmed from a misconception that children had about LOGO.

Do students construct more than one mental model that they apply to different types of recursive programs?

Students do construct more than one model and apply a model depending on its viability for the given program. For example the active model (which could been seen

as a limited version of the copies model) could be applied when the passive flow simply involves control being passed back to previous invocations and no tail-block with any other calculations. On the other hand, students with many misconceptions, tend to apply different non-viable models, depending on what misconception the particular program reveals.

Does prior (iterative) programming knowledge affect construction of mental models of recursion?

Previous research has found that prior experience does not influence understanding of recursion or ability to generate recursive programs. This research found the same with the limited data used to answer this question. So, a student with prior experience would not be more likely to construct a viable mental model than one without. But because students with prior experience probably have viable knowledge about parameter passing, they would be less likely to construct the non-viable return value model. Unfortunately, the majority of the data used in this study did not include information about prior experience and thus this could not be studied in detail.

What misconceptions lead to non-viable mental models?

The root of a non-viable model is a misconception. Students with a step model hold the misconception that recursion is a one or two step process where a condition is tested and one or two steps are executed to reach a solution. These students also hold a misconception about how a computer executes mathematical-type functions (a maths-misconception). A student without a model of computer behaviour who manipulates a recursive program as an algebraic equation or attempts to derive a closed-form formula for that program, has an algebraic model. A maths-misconception together with a misconception about recursive function calls leads to a return-value model (although it might be that a lack of model of how a computer executes embedded function calls could lead to an algebraic misconception). Students who have a viable model of some aspects of recursion but not all, have a magic model. The aspect they don't understand is *magic*ed away and thus they cannot accurately trace the recursive process.

The following mental models and misconceptions were identified:

- The copies model: This is a viable model and thus a student with no misconceptions about recursion will construct this model.
- The looping model: A student with the misconception that recursion is a form of iteration constructs this model.

- The active model: A student with the misconception that the base case signals the end of the recursive process and that the solution is present at the base case will construct this model.
- The magic model: Students who do not have a clear idea of how a recursive program executes and simply *magic* away some detail of the recursive process construct this model.
- The algebraic model: Students with misconceptions about the mathematical nature of recursion construct this model.
- The return value model: Students with misconceptions about parameter passing and function calling and the program stack construct this model.
- The step model: Students with a misconception that recursion is a linear process and without an understanding of function calling construct this model.
- The odd model: Students with a variety of the misconceptions listed above construct odd models.

6.4 Limitations of the research

There are limitations inherent in qualitative research, the most prominent being that the analysis and findings are the researcher's own interpretation of his or her observations of phenomena. While there are many techniques that can be used to ensure sound qualitative research, it is the researcher's argument as laid out in the research report, that must convince a reader of the validity and reliability of the research. The report must make clear all interpretations and provide convincing reasons for those interpretations. Thus a thorough description of the process of the research is as necessary as the discussion of the outcomes. Although a qualitative research study cannot be applied directly in another context, the detailed study provided and the critical processes the research identifies should be relevant in other contexts.

This research is based on the premise that a student's mental model of recursion is revealed by their trace of a recursive program. This is also the premise of previous research identifying mental models of recursion [Kahney, 1989; Kurland and Pea, 1989; Dicheva and Close, 1996; Bhuiyan et al., 1994]. This research further assumes that the instruction: "trace the program's execution showing all workings" is understood by both the researcher and the students to mean: "reveal your mental model", but this shared interpretation may not exist. This was revealed when students simply evaluated

an answer to the recursive call and hence interpreted the instruction as: “what is the answer? ”. A better version of the instruction might have added: “Do not simply give the answer but show how this answer is obtained”.

Another limitation of this research was the lack of control over the recursive questions used from the exams. At the onset of the research it was thought that the majority of the data would come from a specially developed test and that this would be augmented with interviews. The questions in the test and interviews were specifically designed to obtain data suitable for identifying mental models. The data gathered by the test was neither sufficient nor rich enough to identify mental models. This was because only 54 tests were completed and these were not all fully completed. Interviews were not successful, as students were reluctant to volunteer. The few students that did volunteer had viable models of recursion. Thus the data from the students who were interviewed did not add to the understanding of what misconceptions students hold and thus why non-viable models are constructed. While the data gathered from exams was plentiful it did not provide any information about a student’s prior experience nor their own definition of recursion and the questions were not created specifically for this study. The June 2001 exam data was analysed thoroughly before a decision to use exam data in this research was made. It was believed that this data did reveal students’ mental models of recursion, as the questions were the types of programs that students had been exposed to when constructing their models. Also, the two recursive programs had a variety of Scheme syntax, manipulated both lists and numbers, and had different types of recursion: one had only a head-block and the other had head- and tail-blocks.

Over and above the specific questions that were used to identify mental models, is the question of whether it is viable and realistic to identify a student’s mental model from their trace of a recursive program. When a student traces a recursive program he or she is using their cognitive representation of that process to interpret the specific recursive program that they are tracing. In the case of recursion that cognitive representation might be their model of how a computer would execute the program or a more abstract model of recursion. Therefore it is not unfeasible to conclude that it is this mental model that is revealed when a student traces a similar recursive program. A key aspect of recursion is the order of execution of a recursive program, showing the active and passive flow. In order to identify a mental model the order of execution must be determined. A trace is a static representation and thus the order of execution must be deduced. This deduction is the researcher’s interpretation and not necessarily the student’s own. In some traces the order is clear as the student has used arrows to indicate it, but in others the deduction is not as simple and may be incorrect.

The fact that the interpretation was not enhanced with students' own interpretation revealed during an interview as studies by Kahney [1989] and Dicheva and Close [1996] were, is another limitation as it therefore relies too heavily on my own interpretation of traces. But, in comparison to the study by Dicheva and Close [1996], which used students' output of a recursive program to identify mental models, the traces used by this research were more detailed, as student traces showed their understanding of the execution of a recursive program and not simply its output.

The next limitation is whether the process of categorising characteristics of a student's trace allows the identification of a mental model or simply classifies the data into similar groups of traces. Supposing that this research is merely a classification of student traces, I would still argue that it is of value. Whether a student's trace is classified as having certain characteristics or whether it is said to identify that the student has a magic model, these both reveal that the student holds misconceptions and does not have a viable understanding of recursion. The detailed study of each trace that either categorisation or mental model identification requires makes the teacher more aware of what mistakes and misconceptions students hold and thereby could make a teacher more aware and reflective in his or her teaching practice. Thus the objective of this research, to make teachers think about their practice of teaching recursion and enhance that practice, is met.

As this research was exploratory in nature, it has raised questions, and thus there are issues which could be researched further. The next section discusses further research suggestions.

6.5 Suggestions for further research

The exploratory nature of qualitative research often produces new questions that warrant further research. If qualitative research results in the formulation of a new theory, it might be appropriate to do a follow-up quantitative study, measuring the results statistically. In the case of this research project, I believe that the findings should be tested in a classroom situation to demonstrate that this knowledge can inform a lecturers' pedagogy and curriculum, resulting in more effective teaching and learning.

For example a lecturer could become more aware of the first example of a recursive trace that they demonstrate to their class. I would warn against using "factorial", which is the classic recursive example, but could lead to a looping model since the solution can be calculated without a trace of the passive flow. If factorial is used, I would suggest that it is immediately followed by an example where the calculation of the

correct solution requires a detailed trace of the passive flow.

I recommend that a test, such as the one used in this research for the 2002 first year class, be used as a diagnostic testing tool, as Dicheva and Close [1997] have done. These tests can then be analysed, mental models identified and then tutorial groups can be created to deal with specific misconceptions. Students with each type of non-viable model could be tutored in such a way as to challenge that non-viable model and thus facilitate the construction of the copies model. If it is not feasible to create new tutorial groups, the non-viable models could be pointed out with descriptions of why they are not viable. This would make students aware of potential problems and might minimise their occurrence.

In order to determine whether this type of diagnostic teaching is effective, a quantitative study could be conducted. The class could be divided into two groups. The first group could be tutored as students have been in the past, while the second groups' mental models could be identified and their specific misconceptions could then be addressed. A post-test could be administered to determine whether there is a significant difference in the number of copies models identified in either group and thus whether teaching in this way is effective.

It would also be of interest to see if the same test, given to another first year class in another university, would result in the same variety of mental models. This would demonstrate whether models such as the algebraic model are unique to Wits, or whether they are common to other contexts, but have not yet been identified as the programs used by other studies did not bring these models to light.

6.6 Overall conclusion

This research contributes to the field of Computer Science Education, specifically understanding student learning of the concept of recursion. A method for identifying a student's mental model of recursion has been developed. New mental models of recursion have been identified. These are: the *algebraic*, *step*, *return value* and *active* models. The misconceptions that lead to their constructions have been proposed.

The analysis of students' mental models of recursion that this research provides, gives teachers an understanding of how students' misconceptions can impede construction of a viable mental model of recursion. This knowledge can be used to inform pedagogy and curriculum with the objective of facilitating a student's construction of a viable model of recursion.

Appendix A

The Research Instrument

Information Sheet

Dear Computer Science Student

I am conducting my Masters research in the field of Computer Science Education and have proposed to study students' mental models of recursion. A mental model provides an accurate and consistent model of a process. Since research has shown that students have difficulty in learning recursion, it is of interest to discover what mental models of recursion Wits students have constructed after completing the first year course. It is hoped that this research will provide the department with insights into students' understanding of recursion. I also hope to analyse the data that this test provides in order to understand how and why the particular models were constructed. This analysis will take into account the instructional materials provided to first year students and the instructional tactics employed by lecturers in the department. These results will benefit the school as they will provide input for course evaluation and future course design. This should be advantageous to students in the School of Computer Science as learning can be mediated more effectively.

In order to gain knowledge of students' constructions I have developed a data gathering tool. The tool requires participants to provide information about prior programming experience, answer a question about recursion as well trace through some sample Scheme programs. I would like to encourage students to participate in this study, as the data gathered will form my primary data source. Participation is COMPLETELY VOLUNTARY and you will not be in any way discriminated against should you choose not to participate. If you do choose to participate the results will not be used against you in any way and the participation is anonymous. You need not complete ALL the questions, it will be of more use to my research that you complete a few questions in

depth, showing as much of your thought processes as possible.

I would also like to conduct interviews with students in order to gain a deeper understanding of mental models of recursion. This would take place early next year. If you would be willing to participate in an in-depth, video-taped interview early next year please give your name and contact details to me on a separate sheet. All information gathered from interviews is confidential and will not be used for any purpose other than this research. No names will be used in the final report.

Please feel free to contact me if you would like to know more about the research.

Tina Götschi SH2136
tgotschi@cs.wits.ac.za

SECTION 1: Prior Programming Knowledge

Please answer the following question about prior programming knowledge. Place a tick in the column indicating those languages in which you have prior experience and where you gained that experience.

Language	school	university	other course	self-taught
Pascal				
Delphi				
Visual Basic				
C / C++				
Java				
Perl				
LISP				
Scheme				
Other				

QUESTION 2: What is recursion?

In your own words, please explain what you understand recursion to be. Try to give a definition that anyone, not necessarily a Computer Science student would understand. Thus the textbook definition is not required, but rather how you see and understand recursion.

QUESTION 3: Tracing recursive Scheme programs

This question contains 4 sample recursive Scheme programs. They have all be devel-

oped in order to gain information about students' mental models of recursion. Please trace through the program with the program call given. Please show as much of your thought processes as possible, either by using diagrams to represent what you understand to be happening or by writing English statements. Use the back of the page if you need more space. Also write down in a short sentence what the program is achieving.

Program 1:

```
(define prog1
  (lambda (n)
    ( if (< n 1)
         3
         (* 4 (prog1 (- n 2))))))
```

Trace through this program assuming the following call

```
(prog1 5)
```

Program 2:

```
(define helper
  (lambda (lst n)
    (if ( or (null? lst) (= (car lst) n))
        lst
        (helper (cdr lst) n))))

(define prog2
  (lambda (lst n)
    (if (null? (helper (cdr( helper lst n)) n) )
      #f
     #t )))
```

Trace through this program assuming the following call

```
(prog2 '(1 3 8 7 7 8 2 3) 8)
```

Program 3:

```
(define prog3
    (lambda (n)
          (if ( <= n 2)
             1
          (+ (prog3 (- n 1)) (prog3 ( - n 2))))))
```

Trace through this program assuming the following call

```
(prog3 7)
```

Program 4:

```
(define prog4
   (lambda(l)
     (if (null? (cdr l) )
         l
         ( cons (cons (car l) (prog4 (cdr l))) (cddr l)  ) )))
```

Trace through this program assuming the following call

```
(prog4 '(a b c d ))
```

THANK YOU FOR YOUR PARTICIPATION

Appendix B

Trace Methods

The trace methods that were identified were the following:

1. **Boxes** This type of trace used boxes to indicate a procedure or program invocation. A smaller box represented each subsequent invocation with the previous one. Return values are put at the bottom of the box.

2. **Droid** This is a diagrammatic representation of a recursive call as defined by Manis and Little [1995].

3. **Scheme Trace** This style is reminiscent of Scheme's `trace` facility. The program calls are written line by line, with all invocations and their argument values shown. Some students also show the if statements and their results. Some students add further English statements such as "now return #f" or "cdr of the list is null at this point".

4. **Math** This method defines the programs as mathematical functions and then shows how the answer is built inductively.

5. **Arrow** This is similar to Scheme Trace but arrows are used to show the active and passive flows of control.

The most frequently used styles for the 2001 test were Scheme Trace and Boxes. 32 out of the 54 respondents used the same trace method throughout while 22 used different tracing methods.

References

Ben-Ari, M. (1998). Constructivism in Computer Science Education. *SIGCSE Bulletin*, 30(1):257–261.

Ben-Ari, M. and Reich, N. (1996). Recursion: From Drama to Program. *Aspects of Teaching Computer Science*, 7:45–47.

Bereiter, C. and Scardamalia, M. (2000). Beyond Bloom's Taxonomy: Rethinking Knowledge for the Knowledge Age. `http://csile.oise.utoronto.ca/abstracts/Piaget.html`.

Bhuiyan, S., Greer, J., and McCalla, G. (1994). Supporting the Learning of Recursive Problem Solving. *Interactive Learning Environments*, 4(2):115–139.

Bhuiyan, S., Greer, J. E., and McCalla, G. I. (1991). Characterizing, Rationalizing, and Reifying Mental Models of Recursion. In *Proceedings of the 13th Annual Meeting of the Cognitive Science Society*, pages 120–125, Chigago, IL.

Bowman, B. C. and Seagraves, K. (1985). Picturing Recursion. *The Computing Teacher*, 12(7):28–32.

Carbone, A. and Hurst, J. (1999). The student learning experience: characteristics of programming tasks that lead to the poor learning behaviours. In *HERDSA Annual International Conference*, pages 1–17, Melbourne, Australia.

Carbone, A. and Kaasbøll, J. (1998). A Survey of Methods used to Evaluate Computer Science Teaching. *SIGCSE Bulletin*, 30(3):41–45.

Dann, W., Cooper, S., and Pausch, R. (2001). Using Visualization To Teach Novices Recursion. In *Proceedings of the 6th annual conference on innovation and technology in computer science education*, pages 109–112, Canterbury, UK. ACM Press.

Delamont, S. and Hamilton, D. (1984). Revisiting classroom research: a cautionary tale. In Delamont, S., editor, *Readings on Interaction in the Classroom*, pages 3–24. Methuen, London.

Denning, P., Comer, D., Gries, D., Mulder, M., Tucker, A., Turner, A., and Young, P. (1988). Computing as a Discipline. Final Report of the ACM Task Force on The Core of Computer Science. *Communications of the ACM*, 32(1):32–43.

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

Dicheva, D. and Close, J. (1997). Misconceptions in Recursion: Diagnostic Teaching. In *Proceedings of the 6th EUROLOGO Conference – Learning and Exploring with Logo*, pages 234–239, Budapest, Hungary.

Dijkstra, E. (1989). On the Cruelty of Really Teaching Computer Science. *Communications of the ACM*, 32(12):1389–1414.

du Boulay, B. (1989). Some Difficulties of Learning to Program. In Soloway, E. and Spohrer, J., editors, *Studying the Novice Programmer*, pages 283–299. L.Erlbaum, Hillsdale, New Jersey.

du Boulay, B., O'Shea, T., and Monk, J. (1989). The Black Box inside the Glass Box: Presenting Computing Concepts to Novices. In Soloway, E. and Spohrer, J., editors, *Studying the Novice Programmer*, pages 431–446. L.Erlbaum, Hillsdale, New Jersey.

Edwards, D. (1990). Discourse and the Development of Understanding in the Classroom. In Boyd-Barret, O. and Scanlon, E., editors, *Computers and Learning*, pages 186–204. Addison-Wesley.

Fleury, A. E. (2000). Programming in Java: Student Constructed Rules. In *Proceedings of the 31st SIGCSE Technical Symposium*, pages 197–200, Austin, Texas, USA.

Fraenkel, J. R. and Wallen, N. (1990). *How to Design and Evaluate Research in Education*. McGraw-Hill, New York, USA.

Galpin, V. (2001). Course Evaluation Questionnaire. Personal communication.

George, C. (2000). EROSI – Visualising recursion and discovering new errors. In *Proceedings of the 31st SIGCSE technical symposium on computer science education*, pages 305 – 309, Austin, Texas, USA.

Gibbon, P. (1995). A Cognitive Processing Account of Individual Differences in Novice LOGO Programmers' Conceptualisation and Use of Recursion. *Journal of Educational Computing Research*, 13(3):211–226.

Ginat, D. and Shifroni, E. (1999). Teaching Recursion in a Procedural Environment – How much should we emphasize the Computing Model? In *Proceedings of the 30th SIGCSE technical symposium on computer science education*, pages 127–131, New Orleans, Louisiana, USA.

Goldschlager, L. and Lister, A. (1982). *Computer Science: A Modern Introduction*. Prentice Hall, Engelwood Cliffs, New Jersey, USA.

Götschi, T., Sanders, I., and Galpin, V. (2003). Mental Models of Recurion. In *Proceedings of the 34th SIGCSE technical symposium on computer science education*, pages 346–352, Reno, Nevada, USA.

Haberman, B. and Averbuch, H. (2002). The Case of Base Cases: Why are They so Difficult to Recognize? Student Difficulties with Recursion. In *Proceedings of the 7th annual conference on innovation and technology in computer science education*, pages 5–8, Aarhaus, Denmark.

Jaworski, B. (1994). *Investigating Mathematics Teaching*. The Falmer Press, London, UK.

Johnson-Laird, P. (1983). *Mental Models*. Cambridge University Press, Cambridge, UK.

Kahney, K. (1989). What do Novice Programmers Know About Recursion? In Soloway, E. and Spohrer, J., editors, *Studying the Novice Programmer*, pages 315–323. L.Erlbaum, Hillsdale, New Jersey.

Kay, D. (1992). A Balanced Approach to First-Year Computer Science. *SIGCSE Bulletin*, 24(1):15–18.

Kay, D. (1996). Bandwagons Considered Harmful, or the Past as Prologue in Curriculum Change. *SIGCSE Bulletin*, 28(4):55–58.

Kurland, D. and Pea, R. (1989). Children's Mental Models of Recursive Logo Programs. In Soloway, E. and Spohrer, J., editors, *Studying the Novice Programmer*, pages 315–323. L.Erlbaum, Hillsdale, New Jersey.

Levenick, J. (1990). Teaching Recursion Before Iteration. *The Computing Teacher*, pages 12–15.

Levy, D. and Lapidot, T. (2000). Recursively Speaking: Analyzing Students' Discourse of Recursive Phemomena. In *Proceedings of the 31st SIGCSE technical symposium on computer science education*, pages 315–319, Austin, Texas, USA.

Manis, V. and Little, J. (1995). *The Schematics of Computation*. Prentice Hall.

Martins, Y. (2001). Effects of Programming Experience in the Imperative and Functional Paradigms. Honours research report, Department of Computer Science, University of the Witwatersrand, Johannesburg.

McMillan, J. and Schumacher, S. (2001). *Research in Education*. HarperCollins, New York, USA, fifth edition.

Norman, D. (1983). Some Observations on Mental Models. In Gentner, D. and Stevens, A., editors, *Mental Models*. L.Erlbaum, Hillsdale, New Jersey.

Papert, S. (1980). *Mindstorms : Children, Computers and Powerful Ideas*. Harvester Press, Brighton, UK.

Piaget, J. (1964). Development and Learning. In Ripple, R. and Rockcastle, editors, *Piaget Rediscovered*, pages 7–19. Cornell University Press, Ithaca.

Reinfelds, J. (1995). A Three Paradigm First Course for CS Majors. In *Proceedings of the 26th SIGCSE technical symposium on computer science education*, pages 223–227, Nashville, Tennessee, USA.

Sanders, I. and Mueller, C. (2000). A Fundamentals-based First Year Computer Science Curriculum. In *Proceedings of the 31st SIGCSE technical symposium on computer science education*, pages 227–231, Austin, Texas, USA.

Sober, E. (1995). *Core Questions in Philosophy*. Prentice Hall, Engelwood Cliffs, New Jersey, USA.

Sooriamurthi, R. (2001). Problems in comprehending recursion and suggested solutions. In *Proceedings of the 6th annual conference on innovation and technology in computer science education*, pages 25–28, Canterbury, UK. ACM Press.

Trochim, W. M. (2000). Research methods knowledge base. `http://trochim.human.cornell.edu/kb/`.

Turner, A. J. (1991). A Summary of the ACM/IEEE-CS Joint Curriculum Task Force Report. Computing Curricula 1991. *Communications of the ACM*, 34(6):69–84.

Vandenberg, S. and Wollowski, M. (2000). Introducing Computer Science Using a Breadth-First Approach and Functional Programming. *SIGCSE Bulletin*, 32(1):180–184.

Velazquez-Iturbide, J. (2000). Recursion in Gradual Steps (Is recursion really that difficult?). In *Proceedings of the 31st SIGCSE technical symposium on computer science education*, pages 310 – 314, Austin, Texas, USA.

Von Glasersfeld, E. (1992). Questions and Answers about Radical Constructivism. In Pearsall, M., K., editor, *Scope Sequence and co-ordination of secondary School Science Vol 2 Relevant Research*, pages 169–192. National Science Teachers Association, Washington DC, USA.

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

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

Wu, C., Dale, N., and Bethel, L. (1998). Conceptual Models and Cognitive Learning Styles in Teaching Recursion. In *Proceedings of the 29th SIGCSE technical symposium on computer science education*, pages 223–227, Atlanta, Georgia, USA.