# AN ITERATIVE-STYLE APPROACH TO CONSTRUCTING INTELLIGENT TUTORING SYSTEMS IN MATHEMATICS

HYACINTH SAMA NWANA (BSc, MSc)

Doctor of Philosophy

THE UNIVERSITY OF ASTON IN BIRMINGHAM

May 1989

The University of Aston in Birmingham

# An Iterative-Style Approach to Constructing Intelligent Tutoring Systems in Mathematics

Hyacinth Sama Nwana

Doctor of Philosophy

1989

## Summary

Mathematics is highly structured and also underpins most of science and engineering. For this reason, it has proved a very suitable domain for Intelligent Tutoring System (ITS) research, with the result that probably more tutoring systems have been constructed for the domain than any other. However, the literature reveals that there still exists no consensus on a credible approach or approaches for the design of such systems, despite numerous documented efforts. Current approaches to the construction of ITSs leave much to be desired. Consequently, existing ITSs in the domain suffer from a considerable number of shortcomings which render them 'unintelligent'.

The thesis examines some of the reasons why this is the case. Following a critical review of existing ITSs in the domain, and some pilot studies, an alternative approach to their construction is proposed (the 'iterative-style' approach); this supports an iterative style, and also improves on at least some of the shortcomings of existing approaches.

The thesis also presents an ITS for fractions which has been developed using this approach, and which has been evaluated in various ways. It has, demonstrably, improved on many of the limitations of existing ITSs; furthermore, it has been shown to be largely 'intelligent', at least more so than current tutors for the domain. Perhaps more significantly, the tutor has also been evaluated against real students with, so far, very encouraging results.

The thesis thus concludes that the novel iterative-style approach is a more credible approach to the construction of ITSs in mathematics than existing techniques.

Keywords: Intelligent Tutoring Systems, Artificial Intelligence, Computer–Assisted Instruction, Bug Theories, Fractions.

## Dedication

*To my twin brother, Protus Sama Nwana*

*and*

*to the memory of my grandfather, Pa Davidson Mfum Nwana.*

## Acknowledgements

I would like to express my thanks to the following people/organisations:

To Dr & Mrs Nwana and Mr Peter Sama, for everything,

To my supervisor, Dr Peter Coxhead, for his invaluable advice, suggestions, constructive criticisms and his ready service at my every call,

To Dr Brian Gay for providing the necessary hardware, software and a generally good research environment,

To Masoud Yazdani for initially getting me interested in AI/ITS and for his continuous encouragement,

To Prof. Derek Sleeman and Mrs Elizabeth Parkinson for their insightful comments on early drafts of some of the work reported in this thesis,

To various anonymous referees who made valuable suggestions and criticisms of some of the work reported in this thesis when submitted for publication consideration to conferences and journals,

To Dr Neil Simpkins from whom I learnt most of my Prolog,

To Drs Benedict du Boulay, Lindsey Ford, Sara Hennessy, John Self and Mr Rick Evertsz for giving me initial invaluable advice and direction,

To Ken Boden, John Watling and Pam Bartley for putting me in touch with the two Birmingham schools: Ninestiles Mixed Comprehensive School, Solihull and George Dixon Mixed Comprehensive School, Edgbaston; I acknowledge the staff and especially the students of these schools who provided some fascinating protocols priceless to this work,

To Jane Coxhead, Philip Doherty, Rose Jackson & Zaid Jalal for being 'guinea pigs' to the resulting computer-based tutor of this research, To Phillipa Forde for typing parts of this thesis and for her always-available help and friendship,

To my contemporary research students in the Dept. of Computer Science, notably to Nadia Bendjeddou and Robert Thompson, for their general help and friendship, but especially for the various and varied stimulating discussions which has broadened my horizons,

To the Domatob, Fang and the Nwana families for their unending support, encouragement and prayers,

To especially, Gerald Nwana, Peter Ashu, Emmanuel Effiom, Simon Fonebi, Taofeek Folami, Godwin Gwei, Evaristus Mainsah, Josephine Ngong, Ajongwen-Numfor, Jasmine Quansah (who also proof-read this thesis), Martin Shieh, and Peter Wright, but also to all my numerous Cameroonian friends for making my years in Great Britain, not only homely, but memorable,

To the BCS and SSAISB for partly supporting my attending conferences,

To the British Government for the ORS award which partly supported this research,

and last, but by no means least, I acknowledge the financial support of the Cameroonian Government for my entire higher education.

Thanks,
Hyacinth Sama Nwana.

## List of Contents

## List of Tables

## List of Figures

# Chapter 1

# Introduction

## 1.1 Prolegomena

The computer, it has been said, is a once-in-several-centuries invention - a technology which has the potential capability of revolutionising the educational process (Brown, 1985; Papert, 1980; Skinner, 1958; Suppes, 1966). The promise of educational improvement through computers is thus not a new one; educational applications of computer technology have been under development since the early 1960s, i.e. almost 30 years now. By the 1970s, thousands of computer-assisted instructional (CAI) programs had been developed. Generally, however, these did not meet up with the high promises or expectations which educators had for the new medium. Learning via computer was almost always based on fixed presentations of didactic material: such programs were either "electronic 'page-turners', which printed prepared text, or drill-and-practice monitors, which printed problems and responded to the student's solution using prestored answers and remedial comments" (Barr & Feigenbaum, 1982). Furthermore, they had only limited capabilities for adaptive or individualised tutoring and feedback.

To be fair, there were some reasons for the limitations of these early teaching programs. First, the hardware of the time had minimal memory capacity and computational speed. Second, the only theory then available to steer instructional development was behaviour theory (Mandl & Lesgold, 1988): the applicable theory was mainly Skinnerian or neo-Skinnerian behaviourism which poorly matched the cognitive objectives of education. Today, both shortcomings are being overcome. Computer hardware is more powerful: larger memories are continually appearing yearly at even cheaper prices while processing speed has been doubling every two years since 1965 (parallel and/or distributed processing hold even better promises). In addition, there are significantly better languages available for transmitting this power together with improved programming techniques (e.g. logic or object-oriented techniques) which are now available to designers of such instructional software. Expert systems and knowledge engineering technology in particular, and Artificial Intelligence (AI) research in general has played an important role in helping to reveal to the educational fraternity what can be achieved using computers. Further developments are almost certainly likely in this fast-evolving field. More importantly, perhaps, this increased technology has paved the way for other research: for example, machine learning and cognitive modelling. In turn, this has provided useful insights, albeit limited, into human learning in general and this research is still flourishing. Hence, AI and research on human cognition have provided and continue to provide better theories of learning to guide the construction of a new generation of computer-based tutoring programs.

Therefore, one prediction looks secure: that despite the initial set-backs of the 1960s, computers will cause great changes in education. They already have done so, albeit in a limited way: there are already examination halls in American universities where nervous students type answers to multiple-choice questions at computer terminals and anxiously await their grades; there are also experimental classrooms where young children happily and adamantly instruct a computer to draw pictures or play music, and then articulately explain their programs (O'Shea & Self, 1983). The authors also note that there are three reasons for governments, even in times of recession, to encourage the uses of computers in education:

1. Children need to be conscious of the nature and uses of computers in order to be able to survive the present and future technologically-advanced society.
2. Computers can assist with certain administrative burdens, such as the maintenance of student records and scheduling of classes.
3. Computers can help improve the learning process since they are aiding the evolution of better theories of learning as mentioned earlier.

Roecks (1981) lists thirteen uses of computers in education. However, this list is fairly comprehensive and general. This chapter largely concerns computers in educational teaching, i.e. the forms which computer-assisted instructional (CAI) programs take, with particular emphasis on mathematical education. This is because mathematics is the domain to which the main research reported in this thesis applies. The next section examines these forms, and discusses why one of them was chosen as the research area.

## 1.2 CAI in Mathematical Education

Du Boulay (1978) suggests that there are three main forms in which computers are utilised in teaching mathematics. In fact, Roecks (1981) also suggests that all CAI could be roughly divided into the same three categories. They are:

1. Drill and practice
2. Problem solving
3. Tutorial

### 1.2.1 Drill and Practice

Drill and practice is the most common of all the above three forms and contains only two events of instruction. It evokes a response from the learner by asking a question, and provides a feedback generally in the form of knowledge of results. The purpose of this type of CAI, researchers argue, is to provide practice on already learnt skills. The limitations of this form of instruction are well known: for example, some argue that new skills can be taught by this method, but such learning would partake more of 'trial and error' than directed learning and would probably not make efficient use of the learner's time. Thus, this form is generally no longer encouraged by researchers though it is certainly used within other forms of computer-based instruction (e.g. tutorial).

### 1.2.2 Problem Solving (The LOGO Approach)

The problem solving approach attempts to provide learners with opportunities to solve problems within simulated situations (perhaps the name 'problem solving' is not the most appropriate because, as will be observed later on in this thesis, it is also used to mean something else). They resemble 'adventure games' in which in order to complete a task, e.g. traverse a jungle full of wild animals, learners have to suggest solutions to various problems encountered; however, the main goal may be to impart some elementary mathematical skills. Teaching here is through experience (though these experiences are only secondary to their real-life counterparts, which in many cases, e.g. the jungle full of wild animals, are unobtainable anyway); hence, the learning here is often referred to as discovery learning (i.e. learning by doing). Seymour Papert is the most fervent advocate of this technique which is frequently associated with the language LOGO (Papert is LOGO's key author); hence, this approach to using computers in mathematical education is often referred to in the literature as the LOGO approach. LOGO introduces students to the world of geometry through the use of robot 'turtles' and 'turtle graphics' techniques, i.e. the students learn by doing some programming. This approach is based on Piaget's theory of learning which is not surprising considering that Papert was a co-worker of Piaget's for many years. Papert (1980) projects "that computer presence will enable us to modify the learning environment outside the classroom so that much, if not all, of the knowledge schools presently try to teach with such pain, expense and limited success, will be learned as the child learns to talk, painlessly, successfully and without organised instruction". He goes on to conclude that "schools as we know them today have no place in the future". Papert's dream is clearly quite revolutionary: hence, he and his advocates are often referred to in the literature as revolutionalists. His proposed advantages of discovery learning (learning through programming) are (du Boulay, 1978):

1. Programming provides some justification and illustration of formal mathematical rigour.
2. It enables mathematics to be studied through exploratory activity.
3. Programming gives an insight to the mathematical concept.
4. Programming provides a context for problem solving and a language with which the student may describe his/her own problem solving.

However, du Boulay's (1978) investigation into such form of learning mathematics using programming revealed many drawbacks including:

1. Computers are too costly, and will be so for some time to come, to be available for programming to individual students. However, Papert argues that if every child had a computer, computers would be cheap enough for every child to have one.
2. Programming languages are still difficult for students to learn. Even though LOGO is an easier language for communicating with the computer, subtle skills of programming such as planning, debugging, coding, reading, etc. are by no means trivial and have to be learnt. Therefore, there is an obvious difficulty in writing programs of any complexity. Furthermore, concentration on programming aspects was found to obscure what is important mathematically. For example, du Boulay notes an instance where a student was asked to write procedures in LOGO to draw a picture illustrating a mathematical idea, e.g. a fraction pie chart, but she rather concentrated on the details of drawing the picture rather than the underlying mathematical idea to which the picture was supposed to illustrate.
3. Programming languages do not provide good ways of expressing problem solving strategies; they vary significantly with respect to what can be expressed. Papert and his advocates have failed to clearly specify those domains for which the languages they are championing are particularly suitable.

4. Uncoordinated activity by individual students in the sort of setting Papert projects will make education impossible to be assessed in institutions of learning.

### 1.2.3 Tutorial (The Intelligent Tutoring Approach)

Tutorial is the last of the three different forms which CAI takes. A typical procedure for a tutorial program is shown in Figure 1.1. The computer acts as a tutor, i.e. students learn largely by being told. Such tutors aim to provide students with explicit instruction, presenting them with appropriate problems, and offering them informed guidance and feedback on their problems. Hence, the 'hints or remediation' feature (i.e. the shaded rectangle) is important because it can be said to give the system 'intelligence'. Such tutorial systems are termed Intelligent Computer-Assisted Instruction (ICAI) systems or, alternatively, Intelligent Tutoring Systems (ITSs). One of the pioneers of computer tutors (Suppes, 1966) captured clearly the goal of ITSs (though not termed as such at the time) with the following vision: "one can predict that in a few more years millions of school children will have access to what Phillip of Macedon's son Alexander had as royal prerogative: the personal services of a tutor as well-informed and responsive as Aristotle".

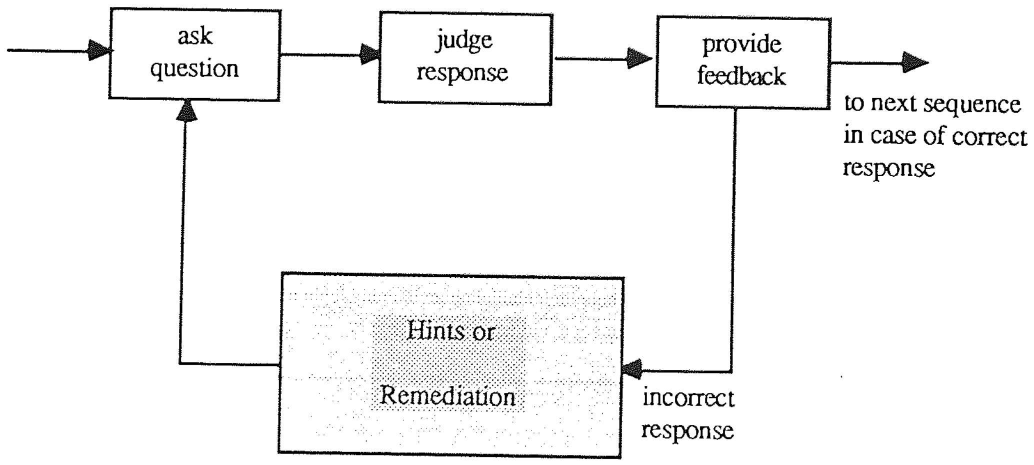


Figure 1.1 - Typical Procedure for a Tutorial CAI

Advocates of this approach (e.g. Suppes) are referred to as reformists, as opposed to revolutionalists: they prefer a gradual improvement (i.e. an evolution) in the present quality of education using techniques like Artificial Intelligence (AI).

### 1.2.4 Discussion

Naturally, the approach Papert champions was bound to be greeted considerable scorn and scepticism as it proposes to radically change the *status quo*. It appears inconceivable that such a radical change is feasible even if it were thought desirable: hence, it is perhaps better to adopt a reformist stance. Besides, there are still many questions that are still unanswered about Papert's approach (some have been highlighted above in Section 1.2.2). The tutorial approach undoubtedly has its pros and cons like the problem solving approach, but it enjoys the privilege of being the closest of the three to current traditional classroom instruction. It is thus the approach most likely to gain widescale acceptance, at least at the present time. Hence, it is the form researched into in this work. Consequently, the work reported henceforth in this thesis mainly concerns CAI of the tutorial type (i.e. Intelligent Tutoring Systems).

## 1.3 Motivation and General Aim of Research

The research reported in this thesis draws on the author's successful M.Sc. thesis (Nwana, 1986) which investigated constructing intelligent tutoring systems for simple arithmetical domains. Before long, the investigation revealed that there was no generally accepted approach to the construction of such systems, even for such structured domains as mathematics. The author's limited knowledge at that time of previous research contributed to making this a surprising result. However, further investigation 'unveiled' some of the difficulty in constructing such systems. Furthermore, two tutors produced as a result of this investigation (the Addition Diagnostic Package (ADP) and the Subtraction Diagnostic Package (SDP)) not only revealed some of the limitations of the approach used to their construction (they are based on a simple 'brand' of the mal-rule approach discussed in a later chapter), but the experience also provided the author with a deeper appreciation of some of the problems involved in designing truly 'intelligent' tutoring systems.

This realisation and appreciation of these problems constituted the main motivation for this research endeavour. Hence, in broad terms, the aim of this research is to investigate an alternative approach to constructing intelligent tutoring systems in mathematics which at least improves on some of the shortcomings of existing approaches. Naturally, the alternative approach has to be evaluated, e.g. by constructing and testing a system based on the approach and comparing it with systems based on current existing approaches in order to reveal the successes and/or failures of the overall research.

## 1.4 Choice of Domain and Language

ITSs have abandoned one of CAI's early objectives, namely that of providing total courses, and have concentrated on building systems which provide supportive environments for more limited topics (Sleeman & Brown, 1982b). Consequently, an important choice that had to be made early on in the research reported here was the selection of a specific topic in mathematics. The domain of the addition and subtraction of fractions was eventually chosen. The reasons for this choice are:

1. The fractions domain is acknowledged to be a difficult topic in the mathematics curriculum (Hasemann, 1981). This is supported by the fact that there is recurring evidence indicating that students' performance with fractions is discouragingly low (Carpenter *et al.*, 1976).
2. Despite the trend away from fractions with the introduction of hand calculators, it will still remain a very important part of the arithmetical curriculum as other mathematical topics (e.g. algebra) rely heavily on it as prerequisite knowledge

(Martinak *et al.*, 1987; Moore & Sleeman, 1987). Therefore, it must be addressed.

3. Research into building ITSs in mathematics has tended to concentrate on integer operations, especially addition, subtraction and multiplication. Some work had already been done in this area (Nwana, 1986) and it was preferred to investigate a new domain.
4. Having looked at the literature, it was realised that, although a small amount of work has been done in this domain - mainly researching into the errors children make in fractions generally, no real tutoring system or attempt to build one existed.
5. Fractions probably provides a more 'complex' domain in that it requires many other subskills in order to execute an addition or subtraction procedure completely and successfully. Some of the skills needed are the addition, subtraction and multiplication of integers, and finding the lowest common denominator. Most of the other diagnostic systems (e.g. ADP/SDP) work on more 'primitive' domains. Therefore, building a tutoring system for the fractions domain promises to provide a more realistic picture of the intricacies of building tutoring systems, since most real domains are complex.

The language chosen for this research is Prolog. The reasons for this choice are threefold:

1. Intelligent tutoring, like AI in general, makes use of rich data structures which can easily be expressed in Prolog.
2. Prolog supports incremental compilation and together with its inherent declarative nature, allows for rapid prototyping.
3. The environment which supports Prolog was the most powerful AI environment available to the author at the start of this work.

## 1.5 Structure of Thesis

The structure of the thesis is as follows:

Chapter 2 introduces the AI subdomain of Intelligent Tutoring Systems (ITS) and the motivation for such tutors. It looks at their history, examines their typical structure and proceeds to review some key ITSs.

Chapter 3 narrows down the view of Intelligent Tutoring Systems to the domain of mathematics as the research reported in this thesis essentially concerns mathematical domains. It critically examines current approaches to the construction of such systems pointing out problems which current ITSs face. It also reviews some other relevant literature.

Chapter 4 reports on some pilot studies that were carried out in order to be able to proceed with the research. These pilot studies yielded some necessary data, produced results which confirmed certain hypotheses and generally provided a much better base to address the central aim of this research, than would have been with only a review of the literature. In brief, it answers some of the questions that the previous chapters left unanswered.

Chapter 5 draws from Chapters 2, 3 and 4, setting out more sharply the central goals of this research endeavour. It also presents the proposed approach to constructing ITSs in mathematics.

Chapter 6 gives an overview of the system which was designed and implemented to test out the proposed approach of Chapter 5; it provides information on how the major modules of the system communicate.

Chapter 7 provides details of the design of the system, i.e. it 'looks inside' all the various modules that constitute the system.

Chapter 8 provides an evaluation of the system. It appraises the system against the specification of the proposed approach, against an independent set of questions to determine how well it lives up to the prefix 'intelligent', and also against real students. It also presents an evaluation of the system's design along with some of its shortcomings.

Chapter 9 presents the conclusions of the overall research endeavour, made partly on the basis of some of the results obtained from the pilot studies reported in Chapter 4, but largely on the evaluation of the system in Chapter 8. This chapter also suggests extensions and further work which could be carried out to improve on the basic system. It concludes with some of the author's viewpoints on a couple of controversial issues in the intelligent tutoring domain.

Appendices A, B and C present the various tests which were given to children to obtain the protocols invaluable to this research which are reported in Chapter 4. Appendices D and E present the systematic errors observed in these tests. Appendix F provides some brief information on how to go about using the system. Appendix G provides some information about the system's modules to anyone who might wish to examine it in more detail. Lastly, Appendix H presents a collection of some of the comments made by students who have used the system.

# Chapter 2
# Intelligent Tutoring Systems: An Overview

## 2.1 Introduction to Intelligent Tutoring Systems

Intelligent Tutoring Systems (ITSs) are computer programs that are designed to incorporate techniques from the Artificial Intelligence (AI) community in order to provide tutors which know *what* they teach, *who* they teach and *how* to teach it. Artificial Intelligence attempts to produce in a computer behaviour which, if performed by a human being, would be described as 'intelligent'; intelligent tutoring systems may similarly be thought of as attempts to produce in a computer behaviour which, if performed by a human being, would be described as 'good teaching' (Elsom-Cook, 1987). The design and development of such tutors lie at the intersection of computer science, cognitive psychology and educational research; this intersecting area is normally referred to as cognitive science (see Figure 2.1). For historical reasons, much of the research in the domain of educational software involving AI has been conducted in the name of 'ICAI', an acronym for 'Intelligent Computer-Aided Instruction'. This phrase, in turn, evolved out of the name 'Computer-Aided Instruction' (CAI) often referring to the use of computers in education. Nevertheless, to all intents and purposes, intelligent tutoring systems (ITSs) and intelligent computer-aided instruction (ICAI) are synonymous. However, though some researchers still prefer 'ICAI' (e.g. Self (1988a) uses it in the title of his recent book), it is being often replaced by the acronym 'ITS' (Sleeman & Brown, 1982b). The latter, which is also the author's personal preference, is certainly gaining more and more support, as confirmed by the fact that there was an international conference on Intelligent Tutoring Systems in Montréal, Canada as recently as June 1988. This preference is motivated by the claim that, in many ways, the significance of the shift in research methodology goes beyond the adding of an 'I' to CAI (Wenger, 1987). However, some researchers are understandably hesitant of using the term 'intelligent', instead opting for labels like Knowledge-Based Tutoring System (KBTS) or Adaptive Tutoring System (ATS) (e.g. Streitz, 1988); Wenger (1987) prefers the label Knowledge Communication Systems. Nevertheless, most researchers appear to be reasonably content with the acronym ITS. This is fine as long as everyone involved with the area understands that the usage of the word 'intelligent' is, strictly speaking, a misnomer. This does not appear to be the case, resulting in some very ambitious goals/claims, particularly in the more theoretical parts of the literature: this also appears to be a valid criticism of the entire AI literature.

The fact that ITS research spans three different disciplines has important implications. It means that there are major differences in research goals, terminology, theoretical frameworks, and emphases amongst ITS researchers. This will be apparent in the subsequent reviews of research in this thesis. ITS research also requires a mutual understanding of the three disciplines involved, a very stressful demand given the problems of keeping abreast with even a single discipline nowadays.

However, some researchers have stood up to the challenge. As a result, a great deal has been learnt about how to design and implement ITSs. A number of impressive ITSs described in this chapter and in Chapter 3 bear testimony to this fact.

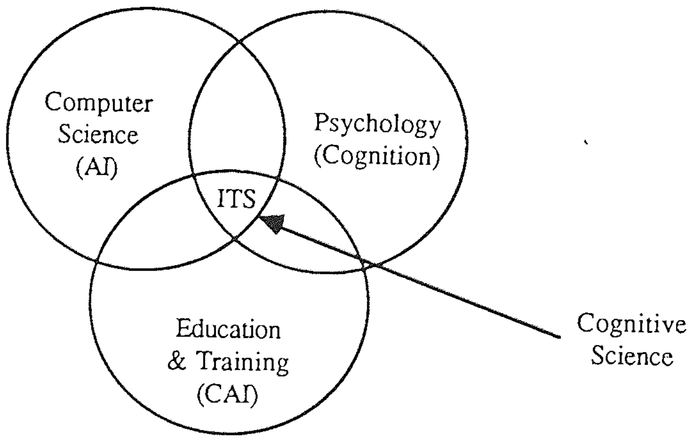


Figure 2.1 - ITS Domains

## 2.2 Motivation

Why do researchers bother to produce such computer-based tutors? There seem to be two main motivating factors namely:

1. *Research needs.* On the pure research level, there is a need to understand more about the processes which contribute to an educational interaction (Elsom-Cook, 1987). Since ITS research lies at the intersection of three main disciplines, it provides an excellent test-bed for various theories from cognitive psychologists, AI scientists and educational theorists. For example, a primary reason why the famous Carnegie Mellon psychologist John Anderson came into the area, was to test out his various theories of learning (Anderson, 1987). Hence, the design of an ITS will contribute to the discovery of more accurate theories of cognition (Burns & Capps, 1988).
2. *Practical needs.* On the more applied level, there are a number of useful results which can be achieved using ITSs which cannot be achieved with human tutors for economic and social reasons (Elsom-Cook, 1987). A primary advantage of ITSs is the possibility for providing one-to-one tutoring. There is a consensus on the view that individual tuition, tailored to the needs of the tutee, is the most effective form of educational interaction, at least for most domains. Bloom (1984) in his comparison of private tutoring with classroom instruction of cartography and probability found that 98 percent of the students with private tutors performed better than the average classroom student, even though all students spent the same amount of time learning the topics. Anderson *et al.* (1985a, 1985b) also recorded a four-to-one advantage for the private tutor, as measured by the amount of time for students to get to the same level of proficiency. Since our educational systems have, of necessity, become geared towards group teaching, many of the advantages of one-to-one tutoring have been lost. ITSs can provide such tuition without necessarily losing the advantages of the group teaching environment (e.g. by providing one ITS per

student in a class), thereby getting the best of both worlds. The ITS could provide immediate feedback to the student on the task being performed. This individualised and immediate feedback is crucial because tutoring is most effective when occurring in direct response to the need of the tutee.

## 2.3 Historical Review

### 2.3.1 Introduction to Review

Computer Assisted Instruction/Learning (CAI/CAL) has evolved considerably since its inception in the 1950s with Skinnerian type 'linear programs'. This has happened despite Skinner having set it off in the wrong direction by his insistence that students' responses could be ignored in linear programs (O'Shea & Self, 1983). The central problem with early systems was that they were unable to provide rich feedback or individualisation, because they were not designed to know what they were teaching, who they were teaching or how to teach it. In order to solve this problem, CAI/CAL systems have evolved over the past three and a half decades into what are now usually termed Intelligent Tutoring Systems (ITSs). Although we may still be far away from truly intelligent tutoring systems, most would agree considerable progress has already been made.

### 2.3.2 From CAI to ITSs: Major Stages

There were some major stages in the metamorphosis of the linear programs of the 1950s into the ITSs of the 1980s (see Figure 2.2). The path has spanned a period of almost four decades. It began in the 1950s with simple 'linear programs' which were based on the principle of operant conditioning. The main proponent of such linear programs was the psychologist B. F. Skinner (1954, 1958). Material which had been selected and arranged to take the student step by step towards the desired behaviour was presented in a series of 'frames'. Most frames had very simple questions (e.g. involving only the filling in of a missing space or two), and the student was told immediately whether the answer was right or wrong. The system proceeded to present the next frame regardless of the correctness of the student's response. To be fair, Skinner held that students should not be allowed to make mistakes because this gives negative reinforcement. If the designer succeeded in this aim, all the responses would be correct and so could legitimately be ignored. Unfortunately, experience showed that such an ideal situation was usually not attainable. The major limitations of linear programs then became glaringly apparent: they did not provide individualisation, which meant that all students, irrespective of their abilities, background, or previous knowledge of the domain, received exactly the same material in exactly the same sequence; neither did they provide feedback, as the students' responses were ignored. This style of CAI has been dubbed *ad-hoc* frame-oriented (AFO) CAI by Carbonell (1970) to stress its dependence on author-specified units of information. Carbonell concluded that "in most CAI systems of the AFO type, the computer does little more than what a programmed text book can do, and one may wonder why the machine is used at all ... when teaching sequences are extremely simple, perhaps trivial, one should consider doing away with the computer, and using other devices or techniques more related to the task" (Carbonell, 1970, pages 194, 201). Overcoming these limitations prompted the chain of events which has culminated in today's ITSs.

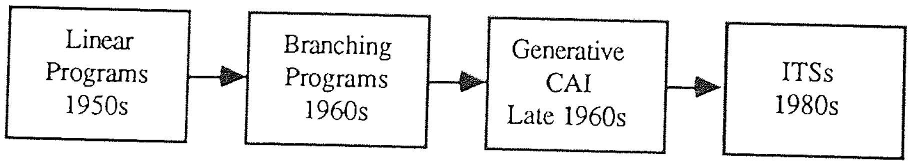


Figure 2.2 - CAI to ITS Metamorphosis

Crowder (1959) overcame some of the limitations of Skinnerian systems by ceasing to ignore students' responses. He proposed using them to control the material shown to the student. The 'branching programs' that resulted still had a fixed number of frames, but were able to comment on a student's response and then use it to choose the next frame, possibly repeating an earlier one. Pattern-matching techniques allowed alternate answers to be treated as acceptable or partially acceptable rather than as totally correct or incorrect as demanded by Skinnerian systems. However, the teaching material became too large to be manageable through straightforward programming and so a special breed of programming languages, called 'author languages', were developed for creating CAI material.

In the late 1960s and early 1970s, 'generative systems' came into being (also called 'adaptive systems'). These emerged from the recognition of the fact that the teaching material could itself be generated by the computer. A generative system has the capability to both generate and solve meaningful problems. In some domains like arithmetic, researchers realised they could do away with all of the pre-stored teaching material, problems, solutions and associated diagnostics, and actually generate them. The potential advantages, if exploited, were enormous. They included drastically reduced memory usage and the generation and provision of as many problems (to some desired level of difficulty) as the student needed. Most notably, Uhr and his team implemented a series of systems which generated problems in arithmetic that were "tailor made" to a student's performance (Uhr, 1969). Suppes (1967) and Woods & Hartley (1971) produced systems with similar abilities. Wexler (1970) describes a system which combines generative CAI with frame-oriented CAI in that the course-author must specify certain question formats. The system generates parameters for these formats and searches the database to determine the correct answer. Nevertheless, a major shortcoming was the restriction to drill-type exercises in domains as well-structured as mathematics. Only parametric summaries of behaviour were used to guide problem generation, rather than an explicit representation of the student's knowledge (Sleeman & Brown, 1982b). Sleeman (1983b) notes that in the initial version of the Leeds Adaptive Arithmetic System (which later evolved to the Leeds Modelling System, LMS), the model of the student consisted merely of an integer to indicate the level of the student's competence.

Generative CAI was the main precursor of ITSs. Although individualisation and feedback had been improved, there was a rather shallow knowledge representation. Yazdani (1986a) notes that "none of these systems has human-like knowledge of the domain it is teaching, nor can it answer the serious questions from the students as to 'why' and 'how' the task is performed". Such sentiments have also been echoed by other researchers (e.g. O'Shea & Self, 1983). Hence, many problems remained unsolved. Sleeman & Brown (1982b) and Hawkes *et al.* (1986) point out that these systems were found to be lacking for such reasons as:

1. They attempted to produce total courses rather than concentrating on building systems for more limited topics.
2. They had severe natural language barriers which restricted users' interaction with them.
3. They had no 'knowledge' or 'understanding' of the subject they tutored or of the tutees themselves; this is sometimes referred to in the literature as the Eliza syndrome. Consequently, they tended to assume too much or too little student knowledge, and they could not conceptualise so as to diagnose a student's misconception within his/her own framework.
4. They were extremely *ad hoc*. Building tutoring systems was not recognised to be a non-trivial task - a task requiring detailed psychological theories of learning and mislearning. Anyone with a knowledge of computing attempted to build a tutor. Consequently, there was little or no cooperation among educators, psychologists and computer scientists in the development phase of the tutors.
5. They tended to be static rather than dynamic. There was little experimentation with systems in order to improve them. Human tutors learn about their students and about the subjects they teach every day, and so should machine tutors.

In response to the problems CAI faced, Self, in his interesting and classic paper, argued that a computer tutorial program should have a representation of *what* is being taught, *who* is being taught and *how* to teach him/her (Self, 1974). Carbonell (1970) argued that a solution to this problem could not be achieved without the use of artificial intelligence (AI) techniques. Jaime Carbonell's important contribution to cognitive science is best summarised in the title of his first-rate 1970 publication "AI in CAI". He wanted to put AI into CAI systems. He dreamed of a system which had a data base of knowledge about a subject matter and general information about language and principles of tutorial instruction. The system could then pursue a natural language dialogue with a student, sometimes following the student's initiative, sometimes taking its own initiative, but always generating its statements and responses in a natural way from its general knowledge. Such a system sharply contrasted with existing CAI systems at the time in which a relatively fixed sequence of questions and possible responses had to be pre-determined for each topic. He constructed an early version of his dream, a classic system he called SCHOLAR (Carbonell, 1970, 1971), but he died before SCHOLAR reached the full realisation of his dream. Carbonell's introduction of AI into CAI marked the beginning of the era of Intelligent Tutoring Systems which therefore emerged to provide answers to the limitations of Generative CAI. (ITSs are alternatively referred to as Intelligent Computer Assisted Instruction/Learning (ICAI/ICAL) systems as mentioned in Section 2.1). It is claimed that ITSs combine AI, psychological models of the student and the expert, and educational theory. The psychological models allow for simulations of student performance and can be experimented upon until they closely represent the behaviour exhibited by the student.

Summarising, providing a truly 'intelligent' system was recognised to be a non-trivial task which needed experts from several other disciplines; a need which could be provided by the AI research community, which contains computer scientists, psychologists and educationalists. Most of the present day work in ITSs is being carried out by AI researchers or enthusiasts, and AI's influence has been so great that Yazdani (1983) concludes that "intelligent tutoring systems are AI's answer to CAL packages". However, it is worth making the point that most so-called ITSs still do not escape the bulk of the criticisms discussed above, so that there is considerable scope for further work. Some of the major differences between ITSs and CAI programs are (or should be):

1. ITSs provide a clear articulation of knowledge for a limited domain.
2. ITSs have a model of tutee performance which is dynamically maintained and is used to drive instruction.
3. The ITS designer defines the knowledge and the inference rules, but *not* the teaching sequence, which is derived by the program.
4. ITSs provide detailed diagnostics of errors rather than simply drill and practice.
5. Tutees can pose questions to an ITS. (This is the main characteristic of 'mixed initiative tutors').

## 2.4 Structure of Intelligent Tutoring Systems

### 2.4.1 General Architecture

Existing ITSs vary tremendously in architecture. In fact, it is almost a rarity to find two ITSs based on the same architecture. This results from the experimental nature of the work in the area: there is yet no clear cut general architecture for such systems (Yazdani, 1986a, 1987). Previously, there was considerable consensus in the literature that intelligent tutoring systems consist of at least three basic components (Barr & Feigenbaum, 1982; Bonnet, 1985) namely:

1. The expert knowledge module
2. The student model module
3. The tutoring module

However, more recent research (Burns & Capps, 1988; Mandl & Lesgold, 1988; Wenger, 1987) has identified and added a fourth component to the list, namely:

4. The user interface module

Figure 2.3 illustrates the general form of an ITS architecture, which will serve as a basis for discussion. It does not represent any particular known system.

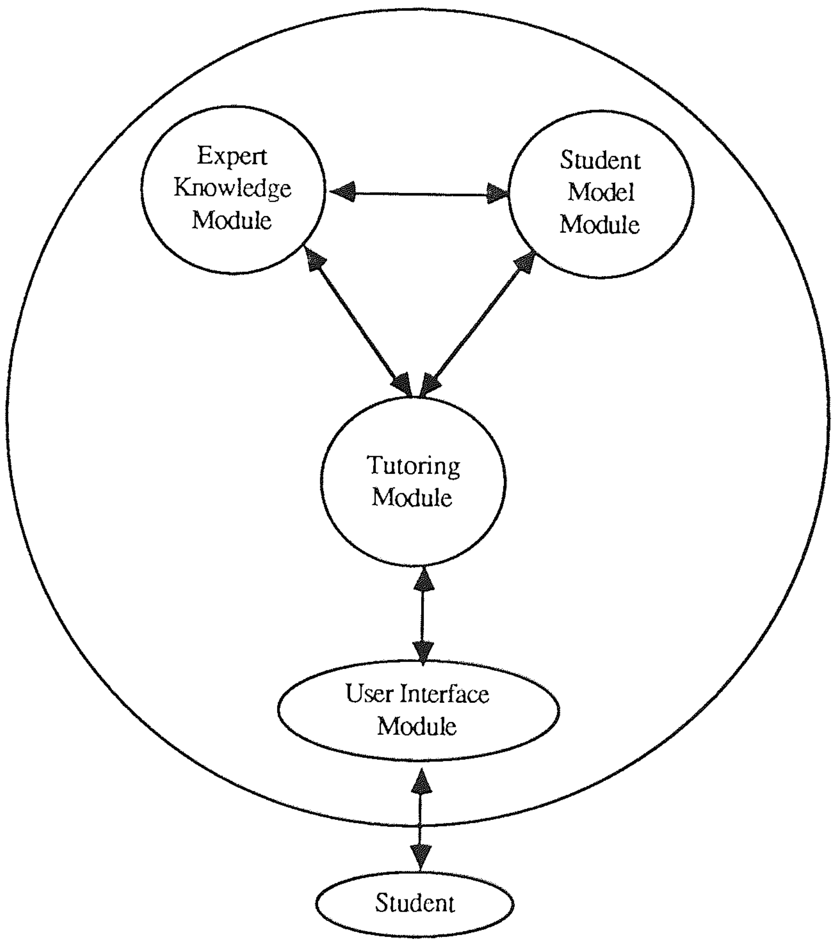


Figure 2.3 - General ITS architecture

*The expert knowledge module* comprises the facts and rules of the particular domain to be conveyed to the student, i.e. the knowledge of the experts. In the transition from CAI to ITSs, such knowledge has been the first aspect of the teacher's expertise to be explicitly represented in systems. It has already been mentioned that in traditional CAI, the expertise to be communicated is contained in prestored presentation material called *frames*, which are designed by the expert teacher and simply displayed to the student under given conditions. Such implicit representation of knowledge has since been recognised to be inadequate: in fact a major lesson learned from all the research on expert systems is that any expert module must have an abundance of specific and detailed knowledge, derived from people who have years of experience in a particular domain. Consequently, in ITSs, much effort is expended in discovering and codifying the domain knowledge, i.e. distilling years of experience into a knowledge representation.

Knowledge elicitation and codification can be a very time-consuming task, especially for a complex domain with an enormous amount of knowledge and inter-relationships of that knowledge. Thus, investigating how to encode knowledge and how to represent it in an ITS remains the central issue of creating an expert knowledge module. In effect, this process aims to make the knowledge stored in this module more explicit. So, in current ITSs, expert knowledge is represented in various ways, including semantic networks, frames and production systems. It must not only include surface knowledge (e.g. the descriptions of various concepts that the student has to acquire), but also the representational ability that has been recognised to be a critical part of expertise. Expert knowledge must include the ability to construct implicit representational understanding from explicitly represented information (Mandl & Lesgold, 1988).

Expert knowledge modules can be classified along a spectrum ranging from completely opaque or 'blackbox' representations, whereby only final results are available (e.g. in tutors like SOPHIE I, reviewed in the next section), to fully transparent or 'glassbox' ones, where each reasoning step can be inspected and interpreted (e.g. SOPHIE III, also reviewed later).

The expert knowledge module or domain expert, as it is alternatively termed, fulfils a double function Firstly, it serves as the source of knowledge to be presented to the student, which includes generating questions, explanations and responses. Secondly, it provides a standard for evaluating the student's performance. For this latter function, it must be capable of generating solutions to problems in the same context as the student, so that respective answers can be compared. The module must also have the ability to detect common systematic mistakes, and if possible identify any gap in the student's knowledge that may be the cause of this. If the ITS is to monitor students in solving problems, the expert module must also be capable of generating sensible, and possibly multiple, solution paths so that intermediate steps can be compared.

Also, in its function as a standard, the expert knowledge module can be used to assess the student's overall progress. To achieve this requires the establishment of some criteria to compare knowledge. This type of comparison is possible only if the knowledge has been explicitly represented. Hence, ITSs differ considerably from traditional CAI programs in that the knowledge in the latter is implicitly represented within its code.

It is also worth recognising that the expert knowledge module by necessity embodies a specific view of the domain - that of the designer. Thus, tutoring can be compromised if the student does not understand the system's instruction or because the system can not interpret the student's behaviour in terms of its own view of the knowledge (Wenger, 1987). Of course, human teachers also have their own views, but they do also have the incredible ability to adapt them accordingly in order to perceive that of the student. This issue touches on the central problem of knowledge representation in AI; a problem which is often said to be the main bottleneck to AI's success. The solution to this bottleneck is particularly relevant to ITSs because of its deep pedagogical implications.

*The student model module* refers to the dynamic representation of the emerging knowledge and skill of the student. No intelligent tutoring can take place without an understanding of the tutee. Thus, along with the idea of explicitly representing the knowledge to be communicated came the idea of doing likewise with the student, in the form of a student model. Most researchers agree that an ITS should have a student model (Gilmore & Self, 1988; Hartley & Sleeman, 1973; Self, 1974, 1987b, 1988b, 1988c; Rich, 1979; Ross *et al.*, 1987; Sleeman, 1985c; Tobias, 1985; Wachsmuth, 1988; Zissos & Witten, 1985). Ideally, this model should include all those aspects of the student's behaviour and knowledge that have possible repercussions on his/her performance and learning. However, the task of constructing such a complete model is not only non-trivial but, probably, impossible; especially considering that the communication channel, which is usually the keyboard, is so restrictive. Human tutors would normally combine data from a variety of other sources, like voice effects or facial gestures. They may also be able to detect other phenomenological factors such as boredom or motivation which are also crucial in learning.

In a recent review, Self (1988c) identified twenty different uses that had been found for student models in existing ITSs. From analysing this list, he notes that the functions of student models could be generally classified into six types:

1. Corrective: to help eradicate bugs in the student's knowledge.
2. Elaborative: to help correct 'incomplete' student knowledge.
3. Strategic: to help initiate significant changes in the tutorial strategy other than the tactical decisions of 1 and 2 above.
4. Diagnostic: to help diagnose bugs in the student's knowledge.
5. Predictive: to help determine the student's likely response to tutorial actions.
6. Evaluative: to help assess the student or the ITS.

In the author's view, Self's preceding list is still far from comprehensive; it could be narrowed down much further, and the student model could be seen to fulfil a double function. On the one hand, it acts as a source of information about the student. On the other hand, it serves as a representation of the student's knowledge. Wenger (1987) also supports this view. The next few paragraphs will attempt to justify this afore-mentioned viewpoint.

In its function as a source of information, it infers unobservable aspects of the student's behaviour from the model. Such an inference could produce an interpretation of his/her actions and also lead to a reconstruction of the knowledge that gave rise to these actions. Such knowledge is vital for the pedagogic component of the ITS as it could be used in either of the six ways noted by Self.

The student model is also likely to be formed out of the system's representation of the target knowledge in the expert knowledge module. Accordingly, the student model can include a clear evaluation of the mastery of each unit of knowledge in the expert module (the function of the student model here is evaluative). This allows the student's state of knowledge to be compared with the expert knowledge module, and instruction would then be biased towards portions of the model shown to be weak (thus, the student model's function here is elaborative). This form of student modelling is referred to as 'overlay' modelling (Goldstein, 1982), because the student's state of knowledge is viewed as a subset of the expert's. Again, this shows how the student model acts as a source of information.

However, incorrect or sub-optimal behaviour does not always result from incomplete knowledge. It could also be due to incorrect versions of the target knowledge. Therefore a more formative student model should also provide explicit representations of the student's incorrect versions of the target knowledge for remediation purposes (clearly, the function of the student model here is diagnostic and corrective). In such a capacity, it serves as a representation of the student's knowledge. This approach to modelling is called the 'buggy' approach (Brown & Burton, 1978); it will be discussed later on in this thesis.

Student models are also expected to be executable or runnable. This allows for exact prediction about a particular student in a particular context (thus, the function of the student model here is predictive). The tutoring module would also be making use of such executable representations for pedagogic purposes (i.e. the knowledge represented in the student model is being used in a strategic way). The student model is also serving here as a representation of the student's knowledge.

In conclusion, student models could be seen to perform two 'super' functions: acting as a source of information about the student, and serving as a representation of the student. In achieving these functions, they act in roles including corrective, elaborative, strategic, diagnostic, predictive and evaluative.

*The tutoring module* is the part of the ITS that designs and regulates instructional interactions with the student. In other architectures, this module is referred to as the teaching strategy or the pedagogic module. It is closely linked to the student model, using knowledge about the student and its own tutorial goal structure to decide which pedagogic activities will be presented: hints to overcome impasses in performance, advice, support, explanations, different practice tasks, tests to confirm hypotheses in the student's model, etc. (Self, 1988b). The tutorial component is thus the source and the orchestrator of all pedagogic interventions. The decisions involved are subtle. The order and manner in which topics are treated can produce very different learning experiences. In tutorial, it is sometimes more effective to let the student flounder for a while before interrupting; sometimes, the student will get stuck or lost if left completely to himself/herself (however, no good human tutor will destroy a student's personal motivation or sense of discovery). Consequently, the tutoring in existing ITSs can be classified along a spectrum ranging from systems that *monitor* the student's every activity very closely, adapting their actions to the student's responses but never relinquishing control, to *guided-discovery learning* systems where the student has almost full control of the activity, and the only way the system can direct the course of action is by modifying the environment. In the middle are *mixed-initiative* systems where the control is shared by the student and the system as they exchange questions and answers. The existence of this spectrum clearly highlights the fact that tutoring is an art that requires great versatility which is still extremely difficult to articulate and represent in an ITS. Nevertheless, some progress is being made, albeit limited.

ITSs also aim at explicitly representing the knowledge found in the tutoring module. This creates the potential to adapt and improve strategies over time (as in the case of self-improving tutors discussed in the next chapter), and for the same strategies to be used for other domains. Once more this is contrasted with traditional CAI systems where such pedagogical knowledge is deeply buried in the various pieces of code that control the tutorial interaction.

*The user interface module* is the communicating component of the ITS which controls interaction between the student and the system, as depicted in Figure 2.3. In both directions, it translates between the system's internal representation and an interface language that is understandable to the student. Because the user interface can make or break the ITS, no matter how 'intelligent' the internal system is, it has become customary to identify it as a distinct component of its own. In fact, it would be a mistake to consider it a secondary component of the ITS for two main reasons. Firstly, when the ITS presents a topic, the interface can enhance or diminish the presentation. Since the interface is the final form in which the ITS presents itself, qualities such as ease of use and attractiveness could be crucial to the student's acceptance of the system. Secondly, progress in media technology is increasingly providing more and more sophisticated tools whose communicative power heavily influences ITS design.

Current ITSs provide user interfaces which, for the input, range from the use of fixed menus with multiple-choice answers to a fairly free treatment of a pseudo-natural language. For the output, they range from the mere display of prestored texts typical of CAI, to the use of fairly complicated generic frames.

Within these two ends of the spectrum there is also a varying flexibility (Wenger, 1987).

Some ITSs are making their interaction with the student more 'user friendly' by substituting pictures and pointing for text and typing. In some, the treatment of text is only supplemented by pictures and graphics; much more cognitive research into the use of such interfaces is still required. However, the most one expects to see, in the near future, is ITSs communicating with the student primarily via graphics, as research in natural language understanding and computer speech recognition/generation are still at such early and primitive stages. In fact, research into user interfaces is just commencing, as they have only recently been acknowledged as distinct parts of ITSs.

### 2.4.2 Other Architectures

Other architectures have been suggested, some largely similar, but others radically different to that depicted in Figure 2.3. For example, there are four components to Anderson's Advanced Computer Tutoring (ACT) ITS architecture (Anderson *et al.*, 1985a, 1985b) (see Figure 2.4). They are:

1. The domain expert (ideal student model): this module contains all the correct rules used for solving problems in the domain.
2. The bug catalogue: this is an extensive library of common misconceptions and errors for the domain.
3. The teaching knowledge (tutoring module).
4. The user interface.

Figure 2.4 - Anderson's ITS architecture

Figure 2.5 - O'Shea *et al.*'s architecture

Hartley & Sleeman, whose 1973 architecture is probably the closest proposal to the general architecture depicted in Figure 2.3, suggest that an ITS ought to have four distinct knowledge bases:

1. Knowledge of the domain (expert knowledge)
2. Knowledge of the person being taught (student model)

3. Knowledge of the teaching strategies (tutoring knowledge)
4. Knowledge of how to apply the tutoring knowledge to the needs of an individual

Hartley & Sleeman's proposals differ from Anderson's inasmuch as they do not give the misconceptions in the domain (the bug catalogue) primary importance but instead introduce the student model as a primary component (Yazdani, 1987). Further, this proposal subsumes the user interface in a more tutoring-oriented module which includes meta-rules that guide the tutoring rules. These differences also reflect the different tutoring philosophies involved in both architectures; Anderson downplays the importance of the student model and, in its place, substitutes two knowledge bases of ideal and buggy representations of knowledge of the domain (see Figure 2.4). Immediately a behaviour is exhibited which indicates an error or bug, the student is nudged to follow the correct path by being presented with the ideal solution.

O'Shea *et al.* (1984) present a five ring model as shown in Figure 2.5. This bears some similarity to the Hartley & Sleeman architecture. However, it also clearly demonstrates how differences in emphasis on student modelling and teaching leads to an architecture which is starkly different from Anderson's (see Figure 2.4). Its components include:

1. Student history
2. Student model
3. Teaching strategy
4. Teaching generation
5. Teaching administration

In this proposal, the explicit representation of knowledge in the domain (expert knowledge), and the common misconceptions in the domain (the bug catalogue), are undermined in favour of teaching skills: hence the introduction of the teaching generation and teaching administration components.

It is thus evident that differing tutoring philosophies place emphases on different aspects of the instructional process (e.g. student modelling), and this in turn leads to different architectures, as in the case of O'Shea *et al.* and Anderson. It is inconceivable that there would be a consensus amongst researchers on a tutoring philosophy; rather, more variations of present philosophies are emerging. Consequently, this has resulted in the numerous ITS architectures in the literature. Naturally, not all these architectures can support a range of tutoring strategies within a given ITS.

Most of these architectures still remain only proposals. Also, almost all the arguments to their support or as to which of them is the 'best' have, disappointingly, been theoretical rather than practical. Until ITSs built around these proposals are evaluated and demonstrated to be educationally worthy, none can claim any superiority over another. Regrettably, a significant fraction of these proposals have not even had systems developed which are based on them. Admittedly, a few have (e.g. Anderson's), but an insufficient number of ITSs based on these architectures have been developed, evaluated and proven 'useful' to warrant such arguments. In fact, due to the experimental nature of the area, even the so-called general architecture of Figure 2.3 should be viewed with some degree of scepticism and should definitely not be seen as a basis for all ITSs. When more developed ITSs become evaluated, as is expected to be the case, it is possible that some additional, or perhaps alternate, set of building blocks might emerge.

## 2.5 Review of some Key ITSs

In this section, some important ITSs are reviewed; an overview chapter of this sort would be incomplete without it. It is by no means meant to be exhaustive: rather, it should augment the previous sections of this chapter. The systems reviewed are either chosen for their historical significance or because they are good examples of systems which display a significant fraction of the intelligent tutoring principles reported in this chapter.

### 2.5.1 SCHOLAR

As the system which successfully launched the new paradigm of intelligent tutoring, SCHOLAR surely deserves a place in any review of the intelligent tutoring systems. Needless to say it was the first ITS to be constructed. It certainly was a revolutionary system when considered in its historical context; most of the then existing ITSs were of the *ad-hoc* frame-oriented (AFO) type. SCHOLAR was created by Jaime Carbonell; this automatically earned him a place in history as founder of ITSs. He used SCHOLAR to launch a new paradigm which he called *information-structure-oriented* (ISO) CAI as opposed to the predominant AFO-type CAI of the time (Carbonell, 1970, 1971). These two approaches correspond in many ways to what are now respectively called traditional CAI and ITS. Because SCHOLAR, like most other ITS projects, evolved, the version described in this review is Carbonell's original version: besides, it is the important one because of its historical significance to ITS research.

SCHOLAR was a pioneering effort in the development of computer tutors capable of handling unanticipated student questions and of generating instructional material in varying levels of detail, depending on the context of the dialogue. It was a mixed-initiative ITS; both the system and the student could initiate conversation by asking questions. Both the program's output and the student's inputs were English sentences. It seems more appropriate to further review it by considering its four components, as defined in Section 2.4.1.

The knowledge in the expert knowledge module is that of the geography of South America, which was represented in a semantic network whose nodes instantiated geographical objects and concepts. Questions like 'Tell me more about Brazil' just invoked a retrieval of facts stored in the semantic network. However, the real power of this representation schema comes by recognising that it is possible to answer questions for which answers are not stored. This automatically relieves the system of the memory problems encountered in anticipating and storing all solutions by traditional CAI systems. For example, one need not store in the semantic network that 'Lima is in South America' provided that the program which interprets the network can make the relevant inference. In other words, the program must know about the attributes concerned, e.g 'location' and 'capital', and in particular, that if x is capital of y and y is located in z then x is in z: this is a rule of inference.

A semantic network representation was chosen because Carbonell thought it to be close to the teacher's conceptualisation of knowledge. By implication, the network also represented the ideal student's conceptualisation (ideal student model): hence, overlay modelling becomes feasible. So SCHOLAR could associate flags with each node of the network to indicate whether the student was thought to know the information represented by that node. More ambitiously, Carbonell proposed to model student errors by introducing small 'perturbations' to the network; this proposal was not followed up in SCHOLAR. Its student modelling was then extremely rudimentary.

SCHOLAR's tutorial strategies were also fairly primitive, consisting mainly of local topic selections. The teacher using it was expected to provide an agenda. Whenever a topic was too general, SCHOLAR generated a subtopic on an essentially random basis. For example, the teacher might specify the topic of 'South

America', and SCHOLAR would select a subtopic, e.g. 'Peru', and then perhaps a sub-subtopic, e.g. 'topography of Peru'. This random element led to somewhat disconnected discussions lacking the systematic development of ideas characteristic of a good tutorial, though it was necessary since SCHOLAR's semantic network had little information about desirable orders of presentations of topics. Nevertheless, suitable relevant tags in the network (from the agenda), could provide SCHOLAR with some reasonable guidance in selecting topics.

SCHOLAR possessed language processing capabilities that were also rather limited. Text was generated by sentence and question templates that was filled up with information from the network. The parsing of student's questions followed the same principle in reverse, while the parsing of student's answers was done by matching keywords from a list dynamically generated from the network for each question. Consequently, SCHOLAR did not understand wrong answers and so could not glean diagnostic information from them.

SCHOLAR has not been widely used except in NLS-SCHOLAR, an intelligent on-line consultant for a text editor (Grignetti *et al.*, 1975). This was partly due to some fundamental limitations such as difficulty of representing procedural knowledge using semantic nets. However, despite all its shortcomings, SCHOLAR introduced many methodological principles that have become central to ITS design, e.g. separation of tutorial strategies from domain knowledge, more explicit representation of knowledge, student modelling, etc. Indeed, SCHOLAR's significance as a milestone for the entire field cannot be over-emphasised.

### 2.5.2 SOPHIE

SOPHIE (a SOPHisticated Instructional Environment) is an ITS, which reflects a major attempt to extend Carbonell's notion of mixed-initiative CAI (introduced in SCHOLAR) for the purpose of encouraging a wider range of student initiatives (Brown & Burton, 1975). It was developed by John Seely Brown, Richard Burton, and their colleagues at Bolt Beranek and Newman, Inc. This project went through successive phases spanning more than five years; the three stages of development of SOPHIE (I to III) incorporate the most intensive attempt at building a complete ITS so far. SOPHIE is a milestone for the field, and hence it well earns its place in this review.

The pedagogic philosophy is different in SOPHIE: it is not so much to imitate a dialogue with a human teacher (as SCHOLAR sought to do) as to provide a reactive learning environment in which the student can try his ideas, have them assessed, and receive advice. Its philosophy is thus 'learning by doing' as opposed to 'learning by being told' as in the case of SCHOLAR. Brown *et al.*, (1982) suggest that computer technology can be used to make experimentation both "easier" and "safer" by simulating environments that capitalise on the motivational value of exploratory problem solving activities. SOPHIE's simulated area of expertise is electronic troubleshooting. Since the components of a simulation can be made faulty, troubleshooting means performing a series of measurements to propose and test hypotheses concerning the location and nature of the fault. This not only gives the student the opportunity to apply a theoretical knowledge of electronic laws, but also to acquire general troubleshooting strategies. In essence, it enables the student to have a one-to-one relationship with an "expert" who helps create, experiment with, and debug his/her own ideas (Brown *et al.*, 1975). In keeping with the objectives of this chapter, SOPHIE will further be reviewed by considering the four components as defined in Section 2.4.1.

The expert knowledge module of SOPHIE comprises of a 'strong' model (simulation) of electronic troubleshooting for the IP-28 regulated power supply and a 'canned' articulate expert troubleshooter which can not only solve problems, but is also capable of explaining its tactics and high level strategies for attacking the problem. For example, the expert can explain 'why' a measurement was made and 'what' logically follows from the measurement obtained.

SOPHIE's tutoring module possesses numerous heuristic strategies for answering and critiquing a student or generating alternative theories to his/her current hypotheses. SOPHIE I's tutorial capabilities were impressive as mentioned in the previous paragraph; however with the evolution from SOPHIE I to SOPHIE III, the strategies were enhanced to become more 'human-like'. This was because its implementors observed that SOPHIE I & II's approaches to problem solving were foreign to humans (Brown *et al.*, 1982). Consequently, SOPHIE evolved from a simulation-based inference system to a more powerful and human-like reasoning system (using qualitative reasoning techniques). By implication, SOPHIE's sophisticated tutoring module requires a similarly sophisticated student model.

By all accounts, the SOPHIE interface demonstrates a very impressive natural language capability; it uses the powerful notion of semantic grammars proposed by Richard Burton. It is robust (handling "nearly all sentences generated by users who have had a few minutes exposure to the system"), efficient ("understands a typical statement in a fraction of a second"), and of some generality ("since the notion of semantic grammars has been successfully applied to other areas besides electronics"); the interface undeniably demonstrates that techniques for processing natural language are sufficiently developed to be usable in ITSs (O'Shea & Self, 1983).

Summarising, SOPHIE's performance as a 'complete' ITS is presently unsurpassed. SOPHIE was actually commissioned by the American Department of Defense and had limited use for on-site job training over the ARPA network for two years; it is no longer maintained (no reason has been given in the literature for this). However, considering that most other prototype ITSs never ever get used after their development, SOPHIE was quite a success story. Probably, SOPHIE's most important contribution to ITS, though, was to establish it as a respectable subarea in the eye of the AI community (Wenger, 1987).

### 2.5.3 GUIDON

GUIDON, an ITS for teaching diagnostic problem solving, was developed by William Clancey and his colleagues at Stanford University. The GUIDON project is also unique as it represents the first attempt to adapt a pre-existing expert system into an intelligent tutor. Probably because it was strongly influenced by SCHOLAR and SOPHIE, it turned out to be one of the most concerted efforts so far at designing an ITS. Like the other two, it went through many stages spanning more than five years during which it yielded many important findings. All of this coupled with the fact that it is built around the most well-known expert system, MYCIN, earns it a place as one of the key ITS projects ever undertaken.

GUIDON's goal is to tutor the knowledge from the famous expert system, MYCIN (Shortliffe, 1976), a medical expert system that suggests treatment for bacterial infections. It attempts to transfer expertise to the students exclusively through case dialogues where a sick patient (the 'case') is described to the student in general terms. The student is then asked to play the role of a physician and ask for information he/she thinks might be relevant to the case. GUIDON compares the student's questions to those which MYCIN would have asked and critiques him/her on this basis; this demonstrates a different tutoring strategy to that of SCHOLAR or SOPHIE. It is easy to infer from the previous sentence that student modelling is largely of the overlay-type. GUIDON also separates its tutorial strategies (comprising 200 rules), which was largely influenced by SOPHIE's, from its domain knowledge. Nevertheless, its natural language capabilities are no where near as sophisticated as SOPHIE's, but certainly improve on SCHOLAR's.

The component which has evolved considerably has been the expert module. The original version (GUIDON 1) was implemented by "reversing" MYCIN's 450 rules. This implementation was ineffectual largely because medical diagnosis is not made "cookbook" style - i.e. medical practitioners do not diagnose diseases by using perfect recall on hundred of medical facts and rules (Clancey, 1982, 1983, 1987). He realised that MYCIN's rules represent 'compiled' knowledge devoid of the low level detail and relation necessary for learning and tutoring. GUIDON 1's failure was largely due to the fact that it would have had to 'decompile' and augment these rules with data and diagnostic hypotheses that the medical practitioner uses implicitly. Thus, MYCIN's rules were reconfigured to separate the strategic knowledge from the domain facts and rules, resulting in NEOMYCIN (Clancey & Letsinger, 1981). This in turn became the new basis around which GUIDON 2 was built, and with some improved teaching strategies, GUIDON 2 has had greater success as a prototype tutoring system than GUIDON 1 (Clancey, 1987).

The GUIDON project provided a fascinating inquiry into the epistemological questions related to intelligent tutoring as well as producing many important findings about designing ITSs. For example, it suggested that an expert system is not a sound basis for tutoring (Elsom-Cook, 1987). More importantly however, GUIDON produced spectacular demonstrations of the field's ability to bring to light fundamental AI research issues (Wenger, 1987).

### 2.5.4 WEST

The WEST coach is a program developed, again, by Richard Burton and John Seely Brown to help students play a game on the PLATO system (PLATO was one of the largest ever CAI projects ever undertaken). It is indeed an intelligent tutor but in view of the implementation of this program in an informal learning environment, the term "coach", originated by Goldstein (1982), appeared more congenial than "tutor" (Burton & Brown, 1977, 1982). WEST was a spin-off from the SOPHIE project; hence, it is still in keeping with the concept of a reactive learning environment central to SOPHIE, but requires much simpler skills. WEST is also the first ever computer coach and it demonstrates how a different emphasis on different components of the ITS (in this case the tutoring module) can produce a radically different ITS, so radical that the term 'coach' is preferred to 'tutor'.

WEST simulates a board game requiring players to travel in a series of moves. The number of spaces for each move are determined by digits on three 'spinners' supplied from a random numbers generator. Players can combine these three digits by using any legitimate mathematical operation including exponentiation, or by using negative numbers, parentheses, etc. The game also has such features as short cuts to the goal; opportunities to 'bump' opponents, forcing them to return to the beginning, and spaces safe from bumping. Although two students can play against each other, they typically play against the machine. At each move, the student's skill in writing algebraic equations is compared to the expert's solution for the same skill. If the two solutions differ, the coach (tutoring component) can intervene and provide the student with helpful hints about how to improve his/her game or make better moves.

WEST's expert knowledge module comprises the simulated board game and an articulate expert which can monitor and evaluate the student's moves. The student modelling technique is largely a simpler version of overlay modelling called differential modelling. This is because, apart from outright arithmetical errors, the student's moves are never wrong; they are just poor. What is important is the difference in comparison between the expert's move and the student's: hence the word 'differential'. WEST's interface is simple as its inputs are mainly arithmetical expressions involving integers (e.g. 1 + 2 * 2) or just plain integers.

However, the component that makes WEST radically different from the previous three systems discussed is its tutoring component. Its main strategy is to encourage skill acquisition and general problem solving abilities by engaging the student in some game-like activity. In effect, the immediate aim is to have fun; skill acquisition and learning is an indirect consequence.

WEST has actually been used in elementary school classrooms. In a controlled experiment, a coached group exhibited "a considerably greater variety of patterns" in the expressions they formed and they even "enjoyed playing the game considerably more than the uncoached group" (Burton & Brown, 1982). These results are quite encouraging: they demonstrate that the coach succeeded in fostering learning without any apparent adverse effect on the fun of the game. Unfortunately, coaching currently appears only to be applicable to trivial domains; it is hard to figure out how it could be used to teach, say, electronic troubleshooting or some non-obvious fraction addition problem.

However, WEST's influence on ITSs has been significant, and it is still a reference for researchers today. Regrettably, it seems it is no longer used. With so many worthless computer games available on the market, it is really unfortunate that a program like WEST, which has actually been demonstrated to be functional, should still remain a laboratory prototype (Wenger, 1987).

## 2.6 Summary

This chapter has introduced the AI subdomain of intelligent tutoring systems and the motivation for ITSs. It also looked at its history: its evolution from CAI. After looking at the structure of a typical ITS, the chapter further reviewed some ITSs, either because of their historical significance or because they best demonstrate the principles of intelligent tutoring.

The chapter also reveals how different emphases on different components of the ITS (because of different philosophies adopted by various researchers) results in contrasting systems; it provides example different architectures to support this. This also rightly reflects the different disciplines which ITS spans.

In the next chapter, the view of the domain of ITSs will be narrowed down to just the domain of mathematics as the main research in this thesis concerns intelligent tutoring in this domain.

# Chapter 3

# Intelligent Tutoring Systems in Mathematics: An Overview

## 3.1 Introduction and Approaches

Mathematics has provided a very suitable domain for ITS research; as Chapter 2 revealed, it played a major role in the evolutionary process from CAI to ITSs. Lovell (1980) explains that there are two main reasons for this:

1. Mathematics is highly structured and its algorithms are well defined, making it easier to concentrate on the features of the ITS itself, rather than those of the domain.
2. Mathematics is an important educational domain because it underpins most of science and engineering.

There are thus many mathematical ITSs. In fact, there are, arguably, more ITSs in the domain of mathematics than for any other domain. However, though many tutoring systems have been implemented for the domain, there still exists no consensus on any approach/approaches for constructing these systems. Nevertheless, after reviewing the numerous existing ITSs for the domain, the approaches to their construction could be grouped under two main headings:

1. The mal-rule approach
2. The model-tracing approach

The next sections of this chapter will look at these approaches in depth, review some key example systems as well as examine some of their shortcomings.

## 3.2 The Mal-rule Approach

The mal-rule (buggy) approach is so termed primarily because mal-rules are central to this approach. (Mal-rules (or bugs) represent the student's misunderstandings of the domain). Systems based on this approach have the characteristic of being diagnostic in nature: they end up diagnosing the student's misconception or give up. According to West, the rationales for the approach are twofold: firstly, that "there is hardly a skill in the teacher's repertoire that is more important than the ability to identify pupils' errors and to prescribe appropriate remedial procedures", and secondly, that errors may be "springboards" for students to understand mathematics (quoted in Borasi, 1986).

However, a more accurate distinction between this and the second approach mentioned above is made by answering Vanlehn's (1988b) bandwidth question: how much of the learner's activity is available to the diagnostic program? Most ITSs in mathematics work on the low end of the information band where only the final state is available to the system, i.e. only the student's answer to a question is available to the system (Burns & Capps, 1988). Such tutors are henceforth referred to in this thesis as mal-rule tutors. Sleeman (1983b) notes that implementors of such ITSs frequently perform the following steps:

1. Analyse protocols from students in the target population solving typical tests and codify their difficulties/misunderstandings. (This may sometimes involve detailed 'clinical' interviews with students).
2. Create databases which include codings of the mal-rules (bugs) observed in Step 1.
3. Implement and use the ITS with students and in particular note student errors which are *not* spotted by the system.
4. Carry out detailed interviews to determine the nature of these errors and code them as additional mal-rules. Steps 3 and 4 are repeated until the system captures the majority of the bugs which occur with the target population.

### 3.2.1 Review of Key Mal-rule ITSs

*BUGGY* (Brown & Burton, 1978) is a mal-rule tutor which provides teachers with practice in diagnosing students' 'bugs' in basic mathematical skills (subtraction) by providing a mechanism for explaining *why* a student is making a mistake rather than simply identifying the mistake. Having a detailed model of a student's knowledge that reveals his/her misconceptions is necessary for successful tutoring: hence the emphasis in this tutor is on student modelling and diagnosis. BUGGY's critical features are a breakdown of skills (e.g. 'add to the next column') into subskills with 'buggy' variants, and an explicit control structure. These constitute a *procedural network.*

The BUGGY game (tutoring) takes the form of BUGGY selecting a problem and choosing 'buggy' versions of one or more subskills in the procedural network to get an incorrect 'student' result. The teacher is now allowed to submit problems of his/her own for BUGGY to answer until he/she deduces the bug. The teacher describes what he/she thinks the bug is (the description is saved but ignored) and a five problem test is given. Here BUGGY provides problems for which the teachers must supply the answers the 'student' would have given. If the teacher is unsuccessful, BUGGY provides more sample problems.

Apparently, student teachers found the game interesting in that what looked initially like a completely random behaviour could be eventually be traced to a consistent buggy procedure. In addition, exposure to the game also improved their ability to diagnose errors (Brown *et al.*, 1977). Brown & Burton (1978) also report similar results when the game was tried in a classroom. They find this result "exciting, since it paves the way for students to see their own faulty behaviour not as being a sign of their stupidity, but as a source of data about their own errors".

*DEBUGGY* (Burton, 1982) is an offshoot of BUGGY which was designed to diagnose up to four or five multiple co-occurring bugs. Unlike BUGGY, it does not calculate the answers of co-occurring bugs in advance; rather, it generates bug combinations dynamically. DEBUGGY basically presents two diagnostic systems: one for dealing with standard tests, and the other for interactive diagnosis (IDEBUGGY). In the former, the aim is to produce an optimum bug set which explains as many of the student's errors as possible. The latter uses the AI paradigm of generate-and-test for diagnosis. IDEBUGGY automatically generates hypotheses as well as problems, if need be, to differentiate amongst hypotheses. It also normally terminates by proposing a diagnosis or gives up.

DEBUGGY is thus a more sophisticated diagnostic system extending the BUGGY model. Burton (1982) also reports that DEBUGGY has been successfully used to diagnose consistent procedural bugs in more than a thousand students.

*ATDSE* (Attisha, 1983; Attisha & Yazdani, 1983) is an acronym for Automated Teacher for Diagnosing Subtraction Errors. The system was designed to diagnose and remediate all known systematic errors for children's subtraction. However, there is a crucial difference in this system as compared to DEBUGGY. In the latter, only the primitive bugs for subtraction are coded: other bugs are generated dynamically from various primitive bug permutations. This complexity is non-existent in ATDSE where all the known subtraction bugs (including those which are primitive bug combinations) are individually coded in the system. So while DEBUGGY can invent bugs, ATDSE is limited to only those represented in its database. Its diagnoses consist of displays of pre-stored 'canned text' explanations of the bugs. ATDAE (Attisha, 1983), EDSMB (Attisha & Yazdani, 1984), ADP/SDP (Nwana, 1986) and HELPERR (Jones & Tuggle, 1979) are similar systems in design to ATDSE.

*LMS* is the acronym for 'Leeds Modelling System' (see Sleeman & Smith, 1981; Sleeman, 1981, 1982a, 1982b, 1983a, 1985a). It is a mal-rule tutor which attempts to infer (predict) models of the student's behaviour in algebra problem solving by using a production rule representation for the rules and mal-rules. That is, given the problems to be worked and the student's answer, LMS is considered to have achieved its goal if the inferred models give the same answers as the student on the problem set. The concept is thus very similar to the correct and 'buggy' rules in BUGGY. However, it does not have the capability of doing any remedial teaching. LMS has since evolved to a system called PIXIE (now that Derek Sleeman, its author, has left Leeds).

*PIXIE* is a workstation-based ITS which presents students with algebra problems, parses and interprets their inputs and evaluates their solutions using sets of predefined rules and mal-rules (Blando *et al.*, 1986; Sleeman, 1987a, 1987b). As in LMS, if a student's answer does not match the answer to the correct rule, the system tries to match the student's answer with those produced by 'mal-rules' in order to infer the buggy model. However, PIXIE's tutoring capabilities have recently been enhanced: for example, it is now capable of making use of some limited intermediate steps, thereby easing the costly diagnostic (tutoring) process (Moore & Sleeman, 1987). Furthermore, PIXIE is now evolving to be the overall name of a family of ITSs which include TPIXIE, an ITS to teach diagnosis of algebra errors (Kelly *et al.*, 1987) and RPIXIE, the remedial version of PIXIE (Martinak *et al.*, 1987). RPIXIE is itself subsumed in FPIXIE, a more general tutor for algebra.

The *QUADRATIC tutor* (O'Shea, 1982a, 1982b) is an ambitious self-improving ITS made up of two components: an adaptive teaching program, and a self-improving component that makes experimental changes using data collected during teaching sessions. However, though not a typical example, it is weakly classified in this review as a mal-rule type ITS since it largely requires only the student's final answer to a question. Nevertheless, its basic design is very different from those of the other tutors previously reviewed. It is largely based on a unique design (O'Shea & Self, 1983) which is reflected in O'Shea *et al.*'s (1984) architecture (see Figure 2.5).

The system attempts to teach the student how to solve quadratic equations using the discovery (learning-by-doing) method. The teaching strategy centres on giving the student carefully chosen examples which increase the likelihood of his/her discovering a particular rule. Both the adaptive teaching component and the self-improving component are expressed in production rules and so the strategy itself is amenable to change. Therefore, the system is capable of carrying out tests and amending the strategy accordingly. O'Shea's attempt at automating a process that is difficult even for humans is an important contribution; most researchers agree that ITSs should improve over time as good human tutors do.

### 3.2.2 Shortcomings of the Mal-rule Approach

It has already been noted that most so-called intelligent tutoring systems still do not escape many of the criticisms levelled at CAI; to be fair, most have actually improved on them. The concept of 'debugging' is the major device used in most mal-rule systems to achieve the diagnosis of errors; DEBUGGY is perhaps the most obvious example. Some of the criticisms levelled at such systems are (Papert, 1980; O'Shea *et al.*, 1988; Ridgway, 1988):

1. The knowledge involved in debugging algorithms reveals rich interconnections between different domains of mathematics. This restricts the value of single domain systems.
2. Some researchers have argued that debugging is an intellectual task which might have great educational value (Papert, 1980) and hence pupils should be

major agents of the debugging process, since they are the ones who have most to gain from it.

3. There are a variety of idiosyncratic methods and approaches that can be used successfully to solve most mathematical problems. Systems such as DEBUGGY neither know about such methods nor can they learn them, because they have no access to how pupils solve problems in the first place. Thus if a range of pupil solutions to subtraction problems is considered, DEBUGGY fares no better than a human teacher, since it will completely fail to debug errors in idiosyncratic methods (Ridgway, 1988).
4. Many systems seem to support a split between an algorithm and its applications. Divorcing these two, Ridgway (1988) argues, "is a device of very doubtful pedagogical value. Subtraction should not be viewed as a self-contained topic which is separate from 'mathematical thinking'. It should be viewed as an integral part of mathematics and mathematical thinking and teaching should focus on extending and strengthening links with other areas of mathematics". O'Shea *et al.* (1988) argue that it is this very divorce that brings about the systematic errors that children make in mathematics.
5. There is still a very high percentage of unexplained errors (~30-40%) (O'Shea *et al.*, 1988).
6. This approach presumes that the novice (student) deviates from the expert (tutor) in some simple way. This is a very common misconception. For example, the student might be viewed as having too few routines, and needs to add more to his/her repertoire; or may be viewed as having some bugs or mal-rules. Ridgway (1988) argues that both these simplistic views are wrong and that there are major differences between novices and experts that are not so readily remedied.
7. The premise that students' errors are systematic is a basic one, but the nature of the systematicity is difficult to define. This leads to the question of bug

stability. Are children's errors transient in nature? Children can be seen to employ two different buggy procedures in solving almost identical problems within a single session (Vanlehn, 1982). Therefore, classifying student's behaviour on some single occasion as being due to various types of buggy behaviour may be woefully misleading.

## 3.3 The Model-tracing Approach

Model-tracing approaches have been around for several years now with its foremost protagonist being the Carnegie Mellon psychologist John Anderson. However, other researchers had earlier suggested such approaches. For instance, Koffman & Blount (1975) report on a system called MALT (MAchine Language Tutor) which can monitor a student's solution to writing a computer program. This technique (model-tracing) is so termed to express the fact that the student is made to follow the system's model quite closely (Anderson, 1984). These tutors guide students through a problem trying to make their steps correspond to those of the ideal student model (see Figure 2.4).

Model-tracing tutors use the expertise of their problem solver to predict the steps the students will take while working on a problem. The problem solver generates all the possible next steps, both correct and incorrect according to the database of rules. They are compared to the student's step and the rule that matches is selected as an interpretation of his/her actions. If the tutor cannot provide a model-driven interpretation of the student's steps, the tutor signals that it does not understand the last input and after a pre-determined number of trials, it simply suggests the next best step according to the ideal model.

Anderson and his colleagues are mainly responsible for the popularity which model-tracing approaches to ITS construction have accrued to date. His earlier work on ACT* (Adaptive Control of Thought) theory (Anderson, 1983) and the "goal restricted production system architecture" of the theory which has been implemented as a more compact model called GRAPES (Anderson *et al.*, 1984b) gives this approach psychological validity. More recently, he has proposed a new production system PUPS (PenUltimate Production System) to replace GRAPES (Anderson & Skwarecki, 1986). These efforts stem from his fervent belief that all pedagogy needs to be rigorously founded in a theory of learning (Anderson, 1988). Anderson *et al.* (1984a) present the cognitive principles on which his tutors are based. He has used the approach in systems for such diverse applications as Lisp and Geometry (Anderson *et al.*, 1985a, 1985b; Anderson & Reiser, 1985; Farell *et al.*, 1984; Reiser *et al.*, 1985). Nonetheless, other researchers not involved with Anderson have constructed, or are still constructing, systems incorporating model-tracing techniques (e.g. Barzilay, 1985; Lantz *et al.*, 1983; Visetti & Dague, 1987).

The main difference between model-tracing tutors and mal-rule tutors lies in the answer to Vanlehn's (1988b) bandwidth question noted in Section 3.2. These ITSs have access to intermediate states: hence, they work with more information (larger bandwidth) than mal-rule tutors. Because the diagnostic module needs as much reliable information as possible about the learner's mental state, bandwidth is critical in designing ITSs. However, the main features of model-tracing are the following (Anderson & Skwarecki, 1986; Anderson, 1987):

1. The tutor constantly monitors the student's problem solving and provides direction wherever the student wanders off path.
2. The tutor tries to provide help with both the overt parts of the problem solution and the planning. However to address the planning, a mechanism had to be introduced in the interface (in this case menus) to allow the student communicate the steps of the planning.
3. The interface tries to eliminate aspects like syntax checking which are irrelevant to the problem solving skill being tutored.
4. The interface is highly reactive in that it does make some response to every symbol the student enters.

### 3.3.1 Review of Key Model-tracing ITSs

The *GEOMETRY tutor* (Anderson *et al.*, 1985a, 1985b), based on the model-tracing approach described above and also on Anderson *et al.*'s (1984a) cognitive principles, tutors high-school students geometry proofs. The key feature of this ITS, like in all Anderson's tutors as previously noted, is its *intelligent monitoring of student activity*: the strategy is to observe the student closely and intervene when he/she makes an error. In this tutor, student's activity is to construct a geometry proof by successively applying geometric theorems, and the tutor compares the student's steps to a large database containing both correct and incorrect problem solving rules. The tutor intervenes when the student applies an incorrect rule, i.e. makes unaccepted or useless moves in the space of possible inferences. However, the success of this tutor may be largely attributable to the insightful use of a proof graph that provides a concrete representation of the concept of proof.

Few ITSs in the literature have resulted in systems complete enough to be evaluated in real instructional settings: the GEOMETRY tutor, however, has been tested with real students. Three high-school students of different aptitudes were involved in a preliminary test. All three reached a good level of proficiency, and even reported liking the previously much-despised geometry after studying with the tutor (Anderson *et al.*, 1985b).

The *SYMBOLIC INTEGRATION tutor* (Kimball, 1982) was one of the first tutors to be developed using a model-tracing paradigm: the bulk of this system was constructed between 1969 and 1972. The goal of symbolic integration is to find a set of transformations that will transform a given symbolic expression into an expression for which the integration is 'known' and obvious. The tutor can either pose problems by choosing from a sequence of example problems in a fixed archive or use problems submitted by the student. The student then indicates an approach for solution such as integration by parts or substitution. In a situation where the student does not know what to do and asks for help, the tutor responds with a recommendation of the best approach. This recommendation is based on a prioritised test of choices among approaches, not on the known solution to the problem as in the case of, say, the GEOMETRY tutor; the expert module, as such, is not actually capable of solving the problem it poses.

This SYMBOLIC INTEGRATION tutor also has the credit of being the first tutor to be labelled 'self-improving' as in the case of the QUADRATIC tutor. This is because the domain expertise can improve in the course of a tutorial session: whenever a student solves a problem in a fewer number of steps than the archived solution, then the student's solution replaces the archived one. In this way, the program's ability to carry out integration improves, and hence its hints to the students may change. However, when tried, Kimball realised that his archived solutions were rapidly replaced by more concise student solutions which turned out to be fortuitous guesses. Despite its limitations, its model-tracing and self-improving features make this tutor quite a success story when observed in its historical entirety, and must have spurred on more ambitious projects such as O'Shea's QUADRATIC endeavour.

*ALGEBRALAND* (Brown, 1985) presents another perspective of model-tracing different from that of the previous two tutors. It is a workbench-type ITS that can be used by students to acquire problem solving skills in high-school algebra. The system, which possesses a very intelligent WIMP-type (Window, Icon, Mouse and Pull-down menus) interface, provides a set of algebraic operators (e.g. combine terms, cancel terms, distribute) that can be successively be applied to an equation until it is solved. The goal here is to free the students from having to perform manually all the calculations associated with different algebraic operations; instead, when solving a problem, students select operations and observe the computer perform them. With the system performing the time-consuming mechanical tasks of symbol manipulation, students are then free to observe the range of applications for an operation, to learn to recognise instances when an operation will be effective, and to understand the limits of its use (Burton, 1988). It is clear from the description so far that the model-tracing approach as exemplified in ALGEBRALAND slightly contrasts that of Anderson's tutors in that it allows for students floundering - something which Andersonian tutors do not tolerate. In any case, as these operations are applied, a search tree is dynamically created and displayed, thereby providing a trace of the problem solving steps taken to solve the problem. This process, called reification, is similar to the proof tree generation process in the GEOMETRY tutor. Brown (1985) notes that such reification of mental processes is very important: it can be used by students, both during problem solving to keep track of their progress, and afterwards, to study the consequences of their errors and to make comparisons with optimal solution paths (Brown, 1983). ALGEBRALAND is thus more of a help system which supports a learning-by-doing paradigm as opposed to the learning-by-being-told style of the GEOMETRY tutor. It aids conceptual understanding of higher-order skills involved in solving problems strategically. Foss observed that students easily acquired such skills, as well as benefited from seeing the consequences of their errors by reviewing the traces created by the system (Foss, 1987).

### 3.3.2 Shortcomings of the Model-tracing Approach

As observed from the above reviews, there appear to be several 'brands' of model-tracing (e.g. the discovery learning-type as exemplified in ALGEBRALAND or the more rigid approach in the GEOMETRY tutor). Anyway, as previously noted, Anderson's is by far the most popular. This is supported by the fact that recent attempts at using model-tracing approaches have produced systems (intermediate systems) more towards the Andersonian types (e.g. Visetti & Dague, 1987). Furthermore, the work of Anderson's team is also amongst the most principled and documented of current ITS research. As a consequence, in listing some of the shortcomings of this approach, there is an obvious bias towards his 'brand' of model-tracing. Model-tracing tutors' drawbacks include:

1. "They seem to incorporate a very dogmatic and authoritarian approach to education" (Yazdani, 1986a). This is because the main driving force behind such tutors is the detection of deviation from the ideal student model (expert problem solver). This has obvious dangers in that the system will reject any other correct approach to solving the problem if it differs from the path in the system. The model-tracing approach currently keeps the student on one of the correct solution paths in the problem; normally the optimal path (Lewis *et al.*, 1987). It is a common misconception to talk about the 'correct' way an expert solves the problem. Ridgway (1988) notes that a major characteristic of experts is that they can solve the same problem in a variety of ways. In short, such tutors are incapable of supporting the variety of idiosyncratic methods that can be used to solve most problems.
2. They feign omniscience (Self, 1988b). But the student could possess strategies that are equally as good or even better than the 'hard-wired' strategy in the ideal student model. Many researchers have called for a more collaborative approach (e.g. Chan & Baskin, 1988). However, it is not always clear that the mechanisms for implementing this have been available.
3. O'Shea & Sleeman (1973) observe that ITSs often have the advantage of an explicit set of tutorial strategies. In model-tracing ITSs, these strategies are often 'hard-wired' in these tutors. Therefore, there is really no explicit representation of knowledge (i.e. the domain-independent and the domain-dependent parts are not clearly delineated). This also leaves no room for improvement by the system itself, though the progress of the student does provide feedback on the teaching process. In brief, such systems cannot be

easily adapted to tutor another domain (Anderson & Skwarecki, 1986): to do so basically requires a total reconstruction of the new system; neither can they learn or improve their strategies over time. There are thus undisputably more benefits to be reaped by constructing systems which have their knowledge explicitly represented. To be fair, this issue is non-trivial: in fact, knowledge representation still remains one of AI's unsolved problems. However, some researchers have had some success, albeit limited, at building self-improving tutors (Kimball, 1982; O'Shea, 1982a, 1982b), largely as a result of explicit knowledge representation.

4. The important educational activity of debugging is taken away by such tutors since this approach, in principle, does not allow for floundering. It may be therefore deemed by some students as being very intrusive and definitely of limited value at teaching debugging skills, i.e. such systems may destroy the student's personal motivation or sense of discovery. (This limitation also applies to the mal-rule approach). This is the central criticism that Papert and his supporters rightly level against ITSs. They claim the approach they champion fully addresses this problem, but as Chapter 1 revealed, there are still many questions about their LOGO technique that requires answering. Indeed, this is a difficult problem; tutoring involves so many subtle skills (e.g. when is it the right time to interrupt?) which good human teachers are gifted with. It is still a major research endeavour how to articulate the knowledge behind such subtle tutoring decisions into ITSs.
5. The style of teaching never changes in such tutors. This is because the rules and mal-rules have merely been 'hardwired' into the system and such designs generally tend to lead to inflexible systems (Ross, 1987). Many researchers, e.g. Barzilay (1985), has stressed the importance of a more flexible style of tutoring, though they seldom indicate how to go about implementing this.

6. The success of such tutors depends heavily on the number of correct rules in the ideal student model as well as the number of mal-rules in the bug catalogue. For instance, the Lisp tutor now contains more than 1200 rules (Anderson & Skwarecki, 1986) more than half of which are mal-rules (Anderson, 1987). Initially, it had 325 production rules for planning and encoding Lisp programs and 475 faulty versions (Anderson *et al.*, 1985a). There are so many rules because of the attempt to encode all the possible paths the student could possibly take. These rules are obtained after painstakingly analysing numerous problem-solving protocols and considerable theoretical analysis of the problem solving domain. One questions how much time and effort ought to go into this analysis phase considering that a handful of mal-rules tend to account for a large percentage of the errors in the domain. Also, the time spent by such systems in diagnosing students' misconceptions could certainly be reduced if the student was made to specify what he/she intended to do next. The uncertainty in what path the student would take is then reduced. Efficiency would also be enhanced with increased flexibility.

7. The main limitations of earlier CAI systems were that they provided neither feedback nor individualisation. Systems such as the Geometry tutor definitely do provide feedback mostly in the form of interrupting either to give advice or for diagnosis when the student wanders off the 'correct' path. However, the system lacks adequate individualisation. This is probably the most severe of all the limitations of the model-tracing approach, at least as exemplified in systems like the Lisp and Geometry tutors. This approach neither supports nor maintains any student models or student history files. Consequently, these tutors do not learn in any way about the student as human tutors do. Therefore such systems would not tutor a weak student any differently from another weak student though their levels of understanding might be significantly different.

Self (1988b) strongly argues in favour of the rehabilitation of student models in intelligent tutoring systems.

8. Some of these tutors tend to do too much for the student. For instance, in ALGEBRALAND, students do not even need to type in anything on the keyboard. It performs all the manual calculations for the student. In fact, in most cases, a response can be generated by pointing to parts of the existing expressions in the current equation window using the mouse; the idea being to concentrate only on aspects that are relevant to the problem-solving skill(s) being tutored. This can be viewed as an advantage or disadvantage depending largely on what view of tomorrow's education one holds. If one is of the radical view that the classrooms of today have no place in tomorrow's schools (Papert, 1980), then this is certainly an advantage. If one rather holds the conventionalist view that present classrooms are here to stay, then many would argue that it is a disadvantage. As Lewis *et al.* (1987) rightly point out, it raises the 'training wheels' issue; what happens when the student leaves such a helpful environment that supports his/her problem solving and has to face the realities of today's pencil-and-paper? It is certainly true that many errors would otherwise occur through slips and mis-strikes of keys or mis-copying of expressions but such (or equivalent) errors also occur in reality when using pencil and paper and students must learn how to recover from them. Admittedly, such help-systems would be very useful in the very early stages of learning a problem solving skill when otherwise the student may be frustrated into giving up due to these 'irrelevant' errors. Perhaps they should therefore be used only at this very early stage and then the student should proceed to more 'lifelike' systems. In summary, it seems that there is a good case for constructing systems which are closer to the reality of the day.

These are only the main criticisms levelled against the model-tracing approach. In fact, some of the shortcomings of the mal-rule technique of Section 3.2.2 also apply to this model-tracing one.

The effect of many of these limitations is to produce a relatively inflexible tutor as the systems strive to anticipate all the possible steps the student might make. Therefore, from the ITS perspective, the model-tracing approach, as exemplified by Andersonian tutors, tends to produce tutors somewhat towards the CAI/CAL end of the spectrum in spite of their apparent cognitive foundation. Wenger (1987) enforces this view as he notes that "...the notion of a compiled ITS corroborates the CAI flavour of this approach to knowledge communication". In fact, Anderson and his colleagues have acknowledged the inflexibility in style of this approach and their recent efforts are enhancing the paradigm's flexibility by proposing a new production system, PUPS, to replace GRAPES on which the current tutors are based (Anderson & Skwarecki, 1986). PUPS is meant to have improved on some of the shortcomings of GRAPES, but Anderson and his colleagues have not yet based any ITS on it; perhaps, it is in the pipeline.

## 3.4 Other Relevant Reviews

Due to the multidisciplinary nature of the domain of intelligent tutoring, there are other areas touched on in the rest of the thesis which requires reviewing; this section presents such relevant reviews.

The first of them reviews the bug theories literature as, they not only form the basis of mal-rule tutors, but they could also provide invaluable clues on how to go about building more intelligent tutors for the domain. The second reviews some of the previous work done in the fractions area; it had been earlier decided that the resulting prototype of this research would be based on some aspect of fractions.

### 3.4.1 Bug Theories

There are two levels of theory involved in research in arithmetical or mathematical errors (Brown & Vanlehn, 1980).

1. Firstly, there are theories of why a student makes the errors he/she does in terms of consistent, systematic but erroneous procedures which explain them. Such procedures are also called bugs or mal-rules and provide the purely descriptive aspect of the bug theory. Many previous researchers put forward such theories (e.g. Ashlock, 1976; Brown & Burton, 1978; Cawsey, 1987; Cox, 1975; Eisenberg, 1976; Ginsburg, 1983; Larkin, 1977; Roberts, 1968; West, 1974; Young & O'Shea, 1981). The fact underlying these theories is well expressed in Ginsburg (1977): "Children make mistakes because they use faulty rules... The faulty rules have sensible origins. Children's mistaken procedures are in fact good rules badly applied or distorted to some degree (page 110)... Errors result from organised strategies and rules. Children's behaviour is meaningful, not capricious (page 129)". Such conclusions are well known to the education fraternity, and can be traced back at least to the 1930s; they contradict the work of Downes & Paling (1958) who blamed errors in arithmetic largely on factors like failing to remember number facts, faulty setting out of exercises, fatigue, carelessness and generally random behaviour. The main claim for fame of these theories is the fact that they form the basis of mal-rule tutors like BUGGY.

   However, the fact that bugs do not suffice to diagnose the application of procedural skills is an important result, which seem to reveal some fundamental limitations of the bug-level analyses (Vanlehn, 1982). This observation helped motivate the development of the more sophisticated theories mentioned next.

2. Secondly, there are theories which accept that errors are caused by bugs, but then attempt to explain why particular bugs occur rather than others (e.g. Brown & Vanlehn, 1980, 1982; Matz, 1982; Young & O'Shea, 1981).

The potential benefits of such second-level theories, if they are successful, are enormous. Not only could it be explained how bugs are caused, but most importantly predictions could be made of what bugs would exist for procedural skills not yet analysed. It would also be possible to automatically generate a list of bugs for a new skill and add them to those already in a database (which might be part of a diagnostic or tutoring system: for instance, Brown & Vanlehn's (1982) theory actually predicted unobserved bugs that appeared later in additional data). Bug remediation could be carried out with more than just a knowledge of what bugs the student has, since such theories could provide a basis for understanding why the student has certain bugs, and with such an understanding, learning environments could be designed to pre-empt and inhibit them. Such theories would also furnish cognitive researchers with insights into cognitive mechanisms like knowledge representation which in turn could lead to the devising of theories of learning and mislearning. Some of the second-level theories are reviewed below.

Matz (1982) suggests that the common errors in algebra arise from one of two main processes:

1. Inappropriate use of a known rule *as it is* in a new situation.
2. Incorrect *adaptation* of a known rule to solve a new problem.

For example, suppose a student knows the base cancellation rule $^{AX}/_{X}=A$. Suppose further that when given the problem $^{(AX+BY)}/_{(X+Y)}$, he/she provides the answer (A+B) as the answer. Matz explains that the child tries to bridge the gap between the known base rule and this unfamiliar new problem. She calls this bridging process *extrapolation*. The student subsequently applies this cancellation rule to every literal instance of it in the expression (as indicated below) producing the error:

$$^{(AX+BY)}/_{(X+Y)} \rightarrow {}^{AX}/_{X} + {}^{BY}/_{Y} \rightarrow A+B$$

She claims that the base cancellation rule which the child knows has been incorrectly *adapted* to solve the new problem and therefore results in an erroneous solution. Her theory, basically, explains both correct rules and mal-rules in terms of children's previous learning and problem solving experience.

Brown & Vanlehn (1980) proposed their generative 'Repair Theory' to explain bugs observed in the subtraction domain. It is meant to provide procedures and constraints that will account for the appearance of the bugs observed and not of others. The theory explains bugs as being due to *repairs* which are constructed when following an impoverished procedure which leads to an *impasse*. In their theory, the formation of bugs is described as a result of "complex, intentional actions reflecting mistaken beliefs about the skill" (Brown & Vanlehn, 1980, page 380). Repairs are normally inventions made by the child so that he/she can continue to execute the procedure to the finish, although in potentially erroneous ways. An impasse is a situation where some current step of the procedure dictates a primitive action that cannot be carried out. Unlike computers, students become inventive when their actions are blocked by a situation their current knowledge cannot cope with. Brown & Vanlehn argue that when a student has unsuccessfully applied a procedure to a given problem, he/she will attempt a repair. In other words, when an impasse in performance occurs, the student attempts to repair his/her performance using weaker methods or he/she seeks help. If the student's chosen metaprocess succeeds in solving the impasse, the student may then abstract the actions needed to deal with the kind of situation just encountered. The problem mainly arises when the students encounter impasses that they cannot resolve with their weak repair methods. Brown & Vanlehn argue that repairs in these situations tend to generate buggy subprocedures. Repairs could therefore be looked on as as temporary patches which are performed so as to allow the execution of the procedure to proceed when it could have been terminated halfway. For example, suppose a student is given the problem 13 - 4. Suppose further that he/she does not know how to 'borrow'. In an attempt to perform the operation on the first column, i.e. 3 - 4, he/she gets stuck as the minuend is less than the subtrahend (he/she arrives at an impasse). The student may therefore perform a repair, e.g. the operation 4 - 3, and hence arrive at the erroneous answer of 11 instead of the correct answer of 9.

Young & O'Shea (1981) found that they could account for a large proportion of the observed errors in children's subtraction, by the deletion of one or more rules, the inclusion of some mal-rules or the replacement of part of the rules' actions in the child's production system (PS). Sleeman (1984a, 1984b) refers this group of bugs as *manipulative* mal-rules, claiming they occur at the time of application: rules are known, but misapplied. As an example, Young & O'Shea argue that deletion of the 'borrowing rule' allows the PS to carry on, but produces wrong answers when the problem requires that a borrow be performed: e.g. 63 - 44 $\rightarrow$ 21.

Vanlehn (1987) has proposed the STEP theory to replace the deletion principle of REPAIR theory by a plausible mechanism for the generation of buggy core procedures. This theory, unlike those mentioned previously, blames the learning process for the bugs children manifest. According to this theory, bugs originate mostly in mislearning and forgetting: that is, in the student's failure to follow a teaching sequence and to organise and retain information, or in the teacher's failure to present sufficiently complete and unambiguous information. SIERRA (Vanlehn, 1987) is a computational model that combines the principles of both theories. The STEP theory proposes certain felicity conditions for the teaching of procedures. ('Felicity conditions' is the name Vanlehn has coined to refer to the conventions that govern teacher-learner discourse). The theory thus provides pedagogical solutions to problems plaguing inductive learning from examples. Vanlehn maintains that this is the form of learning that students apply to build up their repertoire of core procedures in conventional arithmetic curricula. Besides, educators and cognitive scientists have observed that students benefit more from numerous hours spent solving example problems (Woolf, 1988b). In effect, he maintains that misconceptions are themselves often learnt rather than innate or analogised. He argues that textbooks rarely present algorithms as such; they usually present illustrative examples. There are some tutoring maxims behind such examples (Ross, 1987). They are:

1. The examples should be sufficient to learn from.
2. Where there is a choice possible in the implicit algorithm, only one branch should be taught per lesson (one-disjunct-per-lesson) to avoid confusion of goals.
3. In introductory work, nothing should be invisible or unstated.
4. Learning should take place by assimilation rather than restructuring.

Some may deem these as controversial. After all, teachers often present algorithms even if books do not. Vanlehn's SIERRA system attempts to learn from sets of examples which satisfy the above-listed maxims. After each lesson, a subsystem tries to predict the range of bugs that might be observed in students who had reached the same point. In tests with real children, it is claimed that the correspondence between the observed and the predicted bugs were very impressive (Vanlehn, 1987). The maxims above, therefore, constitute the felicity conditions which Vanlehn deems necessary for the tutoring of arithmetical or mathematical procedures.

A crux of his theory is that bugs are likely to occur in a piece of unguided induction that the student performs to make up for a complete (or incomplete) demonstration. Nevertheless, many find it difficult to believe that students learn arithmetic only by induction. This is because this view is too narrow-minded. Wenger (1987) argues that if Vanlehn's theory were accurate, a large number of students would not be able to learn arithmetic at all.

Sleeman (1984a) also suggests a generative mechanism he calls *misgeneralisation* which acknowledges that the genesis of some mal-rules is deep is deep rooted in the learning process. It can therefore be viewed as a step towards a theory which could account for the generation of buggy core procedures like Vanlehn's STEP theory has sought to do. Unfortunately, Sleeman has not yet incorporated this mechanism into a broader learning model, and as a consequence, it is still well away from any plausible generative theory of bugs.

### 3.4.2 Previous Work Done in the Fractions Domain

Work in this area has tended to be mainly researching into the errors children make in the global fractions domain, which therefore includes addition and subtraction operations. Lankford (1972) interviewed 176 students about exercises which involved addition, subtraction, multiplication and division problems. Of the possible 2640 answers, 35% were right, 33% wrong and 21% of the problems were unattempted. He then catalogued the five or six most prevalent errors for each of these fractional operations.

Hershkowitz *et al.* (1980) suggest three categories of errors they observed after having given two open addition exercises to 494 Israeli students of ages ranging from 13 to 15. The exercises were: $^1/_2 + {}^1/_3$ and $^1/_2 + {}^1/_4$. In addition to cataloguing various mistakes under these three categories, they also suggest some cognitive factors as causes of these errors

Sleeman (1987a) in his PIXIE project has also catalogued several mal-rules in various fractional operations which include addition and subtraction.

Shaw *et al.* produced an automatic data analysis program similar to DEBUGGY. Using it, data from many students were analysed, and they discovered 68 bugs in addition of fractions (quoted in Vanlehn, 1988a).

De Gery *et al.* (1985) concentrated on the addition, multiplication and division of two fractions or one fraction and one integer, and the simplification of fractions to arrive at the lowest terms, with the aim of diagnosing the incorrect methods used by students of an average age of 16. These misconceptions were coded as mal-rules and formed the basis of an 'intelligent CAI program' which is still under evaluation. Their system (the latest is called FRACT-2) is still evolving; they have carried out more empirical tests, coded and added more newly observed mal-rules to their database and they ultimately hope to produce an ITS for diagnosing errors in fractions (Dumont, 1988). They also concluded that "it seems that a great part of the students do not think of the meaning of fractions but only use them as some graphic signs, following both true and false rules".

Carpenter, after obtaining answers to the problem $^1/_2 + {}^1/_3$, concluded that "students are not viewing the fraction to be added as representing quantities but see them as four whole numbers to be combined in some fashion" (Carpenter *et al.*, 1976). This conclusion is supported by Hasemann (1981), as well as by the findings of the research programme "Concepts in Secondary Mathematics & Science" (CSMS) based at Chelsea College in London (Hart, 1986; Kerslake, 1986). They conclude that most children do the addition and subtraction of fractions by just applying rules (correct ones as well as mal-rules) without any understanding of why they use these rules.

Evertsz (1982) presents a production system analysis of schoolchildren's fraction subtraction protocols and shows that the majority of errors are systematic, although students still made random as well as number fact errors. Each child is modelled as a production system. Furthermore, he showed that each child's production system generated from half the problems predicts on average 79% of the erroneous answers in the other half (excluding number fact errors). He also presents an analysis of the mal-rules in terms of Repair Theory (Brown & Vanlehn, 1980) and concludes that some, but not all, of the mal-rules can be accounted for in this way. He does not report using such a modelling technique as part of any intelligent tutoring system.

Marshall (1980) presents the Adaptive Diagnostic System (ADS). ADS uses both procedural networks as well as production rules as models for its diagnostic component. Using a provided specification of all the skills and subskills of interest in adding fractions, ADS (a Lisp program) creates a profile of the skills and subskills that an individual may be lacking. It works by generating specific problems to test particular skills and combination of skills. It is thus a probabilistic system. The profile it produces can be used to modify the instructional process.

## 3.5 Summary

In this chapter, the two predominant approaches to developing ITSs in mathematics along with example systems have been discussed and some of their shortcomings have been highlighted. All is clearly not well with ITSs, and although answers to all of these problems are not expected in the immediate future, it is hoped future attempts at building ITSs will tackle at least some of these drawbacks. Lastly, the chapter also reviewed some other relevant literature.

The pilot studies reported in the next chapter was an attempt to address some of these issues raised in this chapter as well as previous ones, so as to shed more light on how to go about constructing an ITS which, at least, improves on some of the limitations of the current two main existing approaches.

# Chapter 4
# Pilot Studies

## 4.1 Introduction to Pilot Studies

The literature review reported in the previous chapters was not deemed sufficient to provide an adequate basis for the entire research. Consequently, it was not clear how to proceed with constructing more intelligent tutors in the domain of mathematics, which is the primary aim of this research. There were four reasons for this 'feeling' of not being ready to proceed, hence necessitating some pilot studies:

1. The review of the literature clearly shows that an ITS requires components such as a bug catalogue or a domain expert (domain problem solver). It was not possible to obtain, 'off the shelf', a bug catalogue for a reasonably 'complex' mathematical domain (e.g. fractions); neither could a good domain expert be constructed without witnessing how real human experts tutor.
2. It was not possible to obtain any of the existing tutors reviewed in the previous chapter to provide some basis for the research, although it is undoubtedly important to have such a base in order to advance the frontiers of research. One way round this, which was adopted in this research, is to construct prototype(s) which simulate state-of-the-art systems and 'play' with these, in order to appreciate their good points and especially their shortcomings; the research could then aim to improve on these limitations.
3. It was felt, after the review, that at least one of the current approaches to constructing ITSs in mathematics was fundamentally questionable. In order to justify this premise, some empirical evidence was necessary. It was also decided that experimenting with a prototype, as mentioned above, would expose more of the approach's inadequacies, thereby also providing another good basis for further research work.

4. It was also considered important to investigate the bugs which children manifest in fractions (the chosen domain to construct the prototype ITS of this research) and the causes of these bugs. These causes were to be as general as possible so as to have wider applicability over the entire domain of mathematics. The reasoning behind this was that, with a general knowledge of errors children make and their underlying causes, ITSs could be designed so as to pre-empt these bugs from occurring in the first place. This led to a bug theoretical study.

For these reasons, it was deemed vital to carry out some pilot studies. As will later be apparent, this turned out to be an extremely valuable exercise.

## 4.2 Design of Diagnostic Tests

A previous section indicated that the first step when building an ITS is to analyse student protocols and to code their misunderstandings or difficulties. This section is devoted to designing the diagnostic tests necessary to uncover errors students make when doing the addition and subtraction of fractions. This was necessary because previous researchers have tended to look at the global domain of fractions as opposed to looking exclusively at these two operations; also their goals were not to produce an intelligent tutoring system. Hence a different perspective on the fractions domain will be presented.

Before the detailed considerations used to design these tests are presented, the reader may find it useful to have pointed out the algorithms used to add or subtract fractions. For problems involving simple fractions only (no whole or loose numbers), the algorithm taught in Britain can be summarized as follows (Evertsz, 1982):

1. If the denominators are not equal, then calculate their Lowest Common Multiple (LCM), i.e. the lowest number into which they will both divide.

2. Calculate the factors for each numerator, and multiply each by its associated factor.
3. Add the numerators, putting the result over the LCM.
4. Cancel if possible.
5. Unvulgarise if necessary.

When mixed numbers are involved, e.g. 2 $^1/_2$, two main algorithms are distinguished in Britain.

One algorithm involves vulgarising the fraction. e.g. 1 $^1/_2$ -> $^3/_2$ and this is normally referred to as the *vulgarising* algorithm. The second method processes the whole numbers and fractions separately and is called the *mixed* algorithm because the numbers stay mixed during the process. Children who use this algorithm need to remember to add the whole numbers while performing the addition operation while children using the vulgarising algorithm will often have to unvulgarise at the end, i.e. convert the vulgar fraction back to a mixed representation. With either algorithm, the child must know how to take care of the 'loose' whole numbers (whole numbers without a fractional part). With the vulgarising algorithm, this will involve changing the whole number into an equivalent vulgar fraction with a denominator equal to the denominator of the other fraction, e.g. 1+$^1/_3$ -> $^3/_3$+$^1/_3$ -> $^4/_3$. With the mixed algorithm it is even easier, e.g. 1+$^1/_2$ -> 1 $^1/_2$. The mixed approach is the more popular one in this country (Evertsz, 1982).

### 4.2.1 Design of Problem Set for Addition of Fractions.

The following factors were used in generating the fractional addition problem set used in this research. The aim of the categorisation below is to improve on the discriminatory power of the test.

1. The Lowest Common Multiple (LCM) of the two fractions is

   (a) obtained by multiplying the two denominators (e.g. $^1/_4$ & $^1/_5$).

(b) the larger of the two denominators (e.g. $^1/_2$ & $^1/_4$).

(c) not any of the above (e.g. $^1/_4$ & $^1/_6$).

(d) either of the denominators i.e. both denominators are same.

2. Each problem falls into one of the categories below.

   (a) First fraction is a mixed fraction.

   (b) Both fractions are mixed.

   (c) Neither is mixed.

3. Some of the problems should involve unvulgarising at the end.

4. Problems should be classified according to whether or not they involve cancelling. If a problem does, then the Highest Common Factor (HCF) is either

   (a) equal to or

   (b) not equal to

   the numerator of fraction being cancelled.

5. Some problems should include a loose whole number as either

   (a) the first or

   (b) the second

   term of the problem.

To ease the students' burden of determining number facts, all of the problems with the exception of one consisted of single digit numbers. The first two factors were used to form a 3 x 4 matrix as set out below (see Figure 4.1). For the reason of testing the phenomenon of bug stability, there were two problems for each square of the matrix giving twenty four problems.

|  | 2a) | 2b) | 2c) |
|---|---|---|---|
| 1a) |  |  |  |
| 1b) |  |  |  |
| 1c) |  |  |  |
| 1d) |  |  |  |

where 1a, 1b, 1c, 1d, 2a, 2b & 2c
are the factors listed above.

Figure 4.1 - Factors Matrix for Addition of Fractions

Hence,

a) there were 8 problems each of type 2a, 2b & 2c.

b) there were 6 problems each of type 1a, 1b, 1c & 1d.

Factors 3 & 4 were divided amongst the problems.

c) 6 problems involved unvulgarising.

d) 6 problems involved cancelling, half of type (4a) and half of type (4b).

Factor 5 lies outside this classification as problems which typify this factor have only one fraction as the one term is loose number, e.g. $1 + {}^1/_2$. For this reason, there were two more problems, one of type 5a and the other of type 5b. This made a grand total of 26 problems. The problems are given in Appendix B of this thesis.

An interesting situation arose during testing, when the author was trying to explain his misconception to a very keen child, who desperately wanted to know his score in the test. His script was analysed while he was around and to explain his misconception, the child was provided with a specific problem which aimed at uncovering his misconception. This is also given in Appendix B. As it transpired, it provides, later on in this thesis, an excellent example to explain one of the implications of building an intelligent tutoring system for fractions.

### 4.2.2 Design of Problem Set for Subtraction of Fractions

The design of the problem set is due to Evertsz (1982). To reduce the burden of number fact determination, all of the fractions consisted of single digit numbers.

The following factors were used to generate the problem set. As with the addition test, the function of the problem categorisation discussed below is also one of improving the discriminatory power of the test.

1. The denominator of the first fraction is

   (a) greater than

   (b) equal to

   (c) or less than

   that of the second.

2. The first numerator is

   (a) greater than

   (b) equal to

   (c) or less than

   that of the second fraction.

3. The LCM of the two fractions is

   a) obtained by multiplying the two denominators, (e.g. $^1/_3$ & $^1/_4$).

   b) the larger of the two denominators (e.g. $^1/_2$ & $^1/_4$).

   c) not found by (a) or (b) (e.g. $^1/_4$ & $^1/_6$).

4. Each problem must fall in one of the following categories.

   a) The first fraction is a mixed fraction.

   b) Both fractions are mixed.

   c) Neither fraction is mixed.

5. Some of the problems should require that a 'borrow' be performed (if a mixed algorithm is used).
6. Some of the problems should involve unvulgarising at the end.
7. The problem should be classified according to whether or not it involves cancelling. If the problem does, then the highest common factor (HCF) is either

   a) equal to (e.g. $^2/_4$) or

   b) not equal to (e.g. $^{10}/_{12}$)

   the numerator of the fraction being cancelled.
8. Some problems should include a loose whole number as either

   a) the first or

   b) the second

   term of the problem.

There were two problems in each cell for the 3 x 3 matrix below (see Figure 4.2) giving eighteen problems. Again, the two problems per cell was for the reason of testing for the stability of bugs.

| D1>D2 | D1=D2 | D1<D2 | |
|---|---|---|---|
| | | | N1>N2 |
| | | | N1=N2 |
| | | | N1<N2 |

where N1 is the numerator of the first fraction,
N2 is the numerator of the second fraction,
D1 is the denominator of the first fraction and
D2 is the denominator of the second fraction.

Figure 4.2 - Factors Matrix for Subtraction of Fractions

The six problems in the second column of the matrix involved calculating a common denominator. This left twelve problems to be divided equally amongst factors 3a, 3b and 3c. Half of the eighteen problems involved borrowing and half did not. Similarly half required unvulgarising, if a vulgarising algorithm was used (not necessarily the same half that required borrowing). Six of the problems required cancelling, half of type 7b and the other half of type 7c.

Factor 8 lies outside this classification since a problem in this category only has one fraction. There were four further problems, two of type 8a and the other of type 8b. Thus there were a total of 22 problems in all. These problems are given in Appendix C of this thesis.

## 4.3 The Pen-and-Paper Tests

### 4.3.1 Method

Subjects The subjects were 74 secondary school students from two separate schools in Birmingham, England. Their ages ranged from 11 to 15 years with the average age being just above 12 years; all were either in their first, second or third year of secondary school. The students were of both sexes and were all from classes of average or greater than average ability arithmetic skills. All the children were of middle or working class origin.

Tests There were three tests involved; the pre-test, the addition test and the subtraction test. The pre-test had 10 miscellaneous but basic problems, well below secondary school standard. It involved questions like 2+3, 5-1, 2×4, etc. (see Appendix A). There were 26 problems in the fractions addition test and 22 in the fractions subtraction test (see Appendices B & C). The numbers vary because of the factors that were taken into consideration when designing the tests. Each student did the pre-test along with either the addition or the subtraction test. A few of the students, who completed the addition or subtraction test in good time, attempted all three tests.

Procedure The children were all seated in their classroom (average group size being 25). They were told by the teacher that they were going to be given some simple exercises to do. Emphasis was placed on the fact that this was not an assessment test and they were assured that the marks were not of prime importance and were not going to be shown to their teachers. They were strongly urged to show as many intermediate steps as possible. No time restrictions were placed on them. They were then presented with the pre-test. Because this was very easy, they took an average of 5 minutes to complete it, by the end of which time they all seemed confident. The purpose of the pre-test was to get rid of any 'test nerves' that still existed; this seemed to work well in practice. Finally, they were presented with either the addition or the subtraction test. Adjacent pupils were given separate tests (addition or subtraction). The intention here was to minimise 'noise' due to copying from neighbours. Previous research has demonstrated that this 'noise' contributes to students' errors (Burton, 1982). A good response was obtained, with some fairly elaborate showing of intermediate steps.

Analysis The students' responses were marked. Those which were incorrect were analysed in detail by hand. Close to 3% of the answers did not lend themselves to analysis due to the fact that they were illegible, and so some information was lost. In other cases, students' mathematical behaviours or protocols were inferred by considering both the students' intermediate and final answers for each problem. Previous work in this domain by other researchers was also consulted during the analysis (Ashlock, 1976; de Gery *et al.*, 1985; Lankford, 1972). Ashlock (1976), Blando *et al.* (1986), de Gery *et al.* (1985) and Cox (1975) have all demonstrated the efficacy of analysing pen-and-paper tests in uncovering students' errors. Finally, systematic errors were inferred for the addition and subtraction operations. A child's errors are said to be systematic if there exists a procedure that produces his/her erroneous answers. In most cases systematic errors are minor deviations from the correct execution of that skill. An example of a systematic error observed in this work is where students, given a problem of the form $^3/_4 + ^1/_2$, always find the LCM correctly but systematically 'forget' to enlarge the numerators. Hence they obtain the erroneous solution, $^3/_4 + ^1/_2 \rightarrow ^3/_4 + ^1/_4 \rightarrow ^4/_4 \rightarrow 1$. If this was consistently observed in a script (enough to justify that they were not 'slips'), then the child was deemed to have this systematic error. The systematic errors observed for addition and subtraction of fractions are given in Appendix D and Appendix E respectively.

### 4.3.2 Results

Table 4.1 - Percentage Classification of All Answers

| | Pre-test | Addition Test | Subtraction Test |
|---|---|---|---|
| Correct | 81 | 36 | 35 |
| Incorrect | 15 | 63 | 61 |
| Not attempted | 4 | 1 | 4 |

Table 4.2 - Percentage Classification of Incorrect Answers

| Error Source | Pre-test | Addition Test | Subtraction Test |
|---|---|---|---|
| Fact | 16 | 2 | 8 |
| Systematic | 68 | 89 | 71 |
| Random | 17 | 9 | 20 |
| Unclassified | 0 | 1 | 1 |

(Note: figures may not sum to 100% because of rounding errors.)

Table 4.3 - Percentage Classification of Bug Stability

| | Addition Test | | Subtraction Test | |
|---|---|---|---|---|
| Stable | 74 | | 64 | |
| Both correct | | 32 | | 24 |
| Both same mal-rule | | 42 | | 40 |
| Unstable | 26 | | 36 | |
| One correct | | 9 | | 27 |
| Different mal-rules | | 17 | | 9 |

(Note: Though not shown, this phenomenon was not confined to a consistent set of students or to a consistent set of problems.)

### 4.3.3 Discussion

The pre-test results (81% correct) were not as good as was anticipated, considering that the questions were so basic. The fact that the children still made errors on such basic primitive operations provided a real cause for concern. If the children had not mastered these skills then it is inevitable that they would encounter difficulties in doing the addition and subtraction of fractions for which the skills tested by the pre-test were pre-requisites. The low percentage of correct answers in the fraction tests, 36% for addition and 35% for subtraction, supports the claim that fractions are to be "recognised as a difficult topic in school mathematics" (Hasemann, 1981).

Tables 4.1 and 4.2 show quite vividly that children's errors are largely systematic. This contradicts the work of Downes & Paling (1958) who blamed errors in arithmetic largely on factors like failing to remember number facts, faulty setting out of exercises, fatigue, carelessness and generally random behaviour; it agrees with more modern research such as that of Ashlock (1976), Brown & Burton (1978), Cox (1975), Eisenberg (1976), Ginsburg (1983) and Young & O'Shea (1981) who have all refuted this claim. However, the results do indicate that factors such as carelessness and generally random behaviour also contribute to student's errors, though not as significantly as systematic errors. The children made a significant number of fact errors (16% for the pre-test) and many systematic errors for primitive operations like addition of integers. In fact, in a similar study, it was found that such fact errors constituted the majority (38%) of the errors observed (Engelhardt, 1977). This definitely makes *post hoc* diagnosis for a domain like fractions more difficult.

Five errors account for 90% of all the systematic errors observed in the addition of fractions. A similar number of errors account for 85% of all the systematic errors observed for subtraction. However, the percentage of random errors for the subtraction of fractions is roughly twice that for addition. Intuitively, this is an expected result since the subtraction process is more difficult than addition. Psychologists can perhaps provide more detailed explanations of this phenomenon.

Bug instability was clearly observed; see Table 4.3. To look for this, two problems of each type were set in both the addition and the subtraction tests. For example, the addition test, which had 26 questions, actually had only 13 problem types. Stability can then be defined as either both answers to a problem type being correct or both exhibiting the same bug. While the majority of pairs of answers (74% for the addition test and 64% for the subtraction test) showed stability, it was interesting to note that in the remaining cases, this was not so. For example, for the problem $^1/_4 + 2$, five students gave the answer $^3/_6$. However, the same five students either gave the correct answer or an answer based on a different mal-rule to the similar problem $^2/_3 + 1$; the expected $^3/_4$ did not appear in any of the five cases. Worse still, bug instability was neither confined to a consistent set of students nor to a consistent set of problems. This phenomenon where often one bug is replaced by another is called bug migration (Vanlehn, 1982). This observation may have crucial consequences in the design of diagnostic/tutoring systems.

## 4.4 FRACTIONLAND

### 4.4.1 Purpose and Capabilities of FRACTIONLAND

FRACTIONLAND is a prototype system which re-implements the capabilities of a rather large number of systems often referred to as 'intelligent tutors' in the domain of mathematics. Its domain is the addition of fractions (note that the number of fractions to be added is limited in this tutor to two). The reasons for its development were threefold.

1. It was aimed to provide a rough and rudimentary idea of the state-of-the-art in the domain of mathematics except for a handful of tutors (e.g. Anderson's GEOMETRY tutor and ALGEBRALAND). Nonetheless, it roughly re-implements the capabilities of mal-rule systems like BUGGY, ATDSE/ATDAE, etc. (see Chapter 3). Having a good understanding of the state-of-the-art is a crucial base for advancing the frontiers of research. Ideally, it should be possible to 'play' with some of the existing systems in order to appreciate their weaknesses and their mediocre as well as their good aspects. Most ITS researchers would agree that even the most written-about systems in the literature are still unavailable today to others in the domain to research on. (For this reason, a main conclusion at an ITS meeting was that a UK ITS resources centre should be started up (Ford & Yazdani, 1987)). Without such a centre, the only other alternative left is to 're-invent the wheel' and re-implement these systems to research on.
2. A decision had earlier been made to develop the eventual tutor in Prolog. With no previous hands-on experience with Prolog, it was deemed a good idea to construct, at least, a simpler and more rudimentary system and in so doing gain some invaluable experience working with Prolog as well as the environment which had been chosen to develop the eventual tutor. It turned out to be a very valuable exercise in this respect.

3. FRACTIONLAND provided something to compare and contrast with the eventual tutor and thus provide a reasonable 'baseline' for the assessment of the success of the entire research endeavour as a whole.

FRACTIONLAND is about 90K of source code. It is based on the mal-rule approach and incorporates the primitive and compound bugs found from the empirical tests to be the most frequent bugs for addition of fractions (see Appendices D and E). It provides the following facilities to the user:

- The user can provide a fraction problem and the system provides the answer.
- The user can explore some of the bugs in the domain (mainly those mentioned previously).
- The user can provide a problem and the answer for the system to check if it is correct.
- The system can provide the user with the chance to play the game of BUGGY (Brown & Burton, 1978).
- The system can interactively diagnose a solution by generating problems to distinguish amongst possible hypotheses, if there are several. This is similar to IDEBUGGY (Burton, 1982).
- The system can diagnose errors in a file of addition of fraction problems and solutions by producing an optimum set of bugs which also take into account features like subsumption (where one bug subsumes another). Once more, this is akin to Burton's (1982) off-line version of DEBUGGY.

### 4.4.2 Lessons Learned from FRACTIONLAND

As a vehicle or tool for experimental research, it clearly provided many lessons to guide the building of a more intelligent tutor. In addition to the problems already identified with ITSs based on mal-rule approach (see Chapter 3), some others identified after 'playing' with this prototype are listed below.

1. Mal-rule ITSs provide environments where the initiative is completely taken by the tutor and the user's interaction is reduced mostly to monosyllabic answers to the system-generated questions. This is in stark contrast to Papert's (1980) philosophy of providing the user with almost complete control of the initiative. However, many researchers have argued for a compromise - a rather more mixed initiative interaction with the system and the user taking the initiative when necessary (e.g. Carbonell, 1970).
2. Some of the textual explanations shown to the users of the bugs observed are so terse that they will be of little use to them. On the other hand, if explanations are made very long-winded, the system tends to resemble the early frame-based systems. It was realised that a compromise is quite difficult to achieve.
3. Even with the best of explanations of the bugs diagnosed, the system still did not provide much help to the user on how to go about getting the problem correct next time round. One of the main reasons for this is that the fractions domain is inherently more 'complex' than, say, integer subtraction, on which systems like BUGGY (Brown & Burton, 1978), DEBUGGY (Burton, 1982) and ATDSE (Attisha, 1983; Attisha & Yazdani, 1983), are based. Diagnosing a bug as "the child adds both numerators and both denominators to obtain the numerator and denominator of the answer" achieves little in helping the user to solve the problem correctly. In short, most of these systems diagnose without remediating. Self (1987b, 1988b) argues that there is no point diagnosing if remediation cannot be done.
4. It was not unusual to see the interactive diagnostic part of the system generating 3 or 4 extra problems in its attempt to diagnose a single problem. This can be very time consuming, boring and of questionable educational value. This is especially noticeable in ADP/SDP (Nwana, 1986).
5. One of the main criticism of ITS's precursor, CAI, is that it performed neither feedback nor individualisation. The ITSs mentioned so far do provide some

feedback, mostly in terms of diagnostic error messages generated but rarely do they provide adequate individualisation. This is clearly because they contain no model whatever of the student's state of knowledge. BUGGY, DEBUGGY, and ATDSE have no explicit student models which could be used to drive individualised instruction.

6. Such ITSs tend to be very monotonous and uninteresting as the style of tutoring never changes. They just usually generate problems which they present to the student to provide answers to. Then they proceed to diagnose the answers. A more flexible tutoring style would certainly be of more educational value (Barzilay, 1985).
7. Such systems ultimately end up being of little or no use to real students due to some of the preceeding reasons. In fact, many end up being of more use to student teachers as in the cases of BUGGY (Brown & Burton, 1978), ATDSE (Attisha, 1983; Attisha & Yazdani, 1983) and ADP/SDP (Nwana, 1986). Surely, real tutoring systems demand the shifting of focus from the student teachers to the student themselves.

Doubtless, other criticisms could be made, although the main ones have been either highlighted above or in Chapter 3. In conclusion, implementing FRACTIONLAND, even though it was 'quick and dirty', was a very fruitful exercise. Lessons were learnt which provide guidelines on how to go about building tutors, at least in the domain of mathematics. In addition, it certainly provided valuable hands-on experience, albeit limited, on how to build intelligent tutors. Lastly, the eventual tutor can be compared and contrasted with FRACTIONLAND in order to appreciate its successes and/or failures.

## 4.5 Bug Theoretical Study

The main motivation behind this aspect of the pilot study is the fact that to construct an ITS for a domain, the designer needs to acquire a deeper knowledge of that particular domain, for instance, knowledge of the errors that children manifest in the domain (Anderson *et al.*, 1985a; Visetti & Dague, 1987), and perhaps more preferably, knowledge of the underlying causes of these errors (Brown & Vanlehn, 1980; Sleeman, 1984). The former could be incorporated in a bug catalogue and used in an ITS as in the cases of BUGGY and PIXIE; the previous chapter has already addressed this. However, knowledge of the latter may even be more important because it would, theoretically, be invaluable in constructing the pedagogic component of the tutoring system and hence devising ITSs which prevent these bugs from occurring in the first place. This section titled 'bug theoretical study' was an attempt to address this latter issue.

### 4.5.1 The Most Frequent Bugs Observed

Below, are the five most frequent bugs observed for the addition and subtraction operations starting with the most frequent. They also constituted 90% and 85% of all the bugs observed in the addition and subtraction tests respectively.

#### Addition of Fraction Errors

1. The child adds both numerators and both denominators to obtain the numerator and the denominator of the sum respectively. This bug alone constituted 73% of all the bugs observed for the addition test; in agreement with Lankford (1972) when he notes that "this is the most prevalent practice" observed.
   Example: $^3/_4 + ^1/_5 \rightarrow ^4/_9$.
2. The child adds all the values of the first term (which may be fractional, whole or both) and records this value. The same process is done for the second term. Then, the smaller value is put over the larger to provide the sum.
   Example: $1\,^3/_4 + ^4/_7 \rightarrow ^8/_{11}$.
3. When either or both of the terms are mixed, the whole numbers are added to the sum of the numerators and this value is recorded. The denominators are also

summed. The sum of the fraction is taken to be the former value over the latter. Example: $1\ ^2/_4 + 1\ ^1/_2 \rightarrow {}^{(1+2+1+1)}/_{(4+2)} \rightarrow {}^5/_6$.

4. The child does not cancel in the final answer. Example: $^4/_8 + {}^2/_8 \rightarrow {}^6/_8$ (should cancel $\rightarrow {}^3/_4$).
5. The pupil multiplies instead of adding. Example: $^1/_4 + {}^1/_6 \rightarrow {}^1/_{24}$.

### Subtraction of Fraction Errors

1. The child first finds the difference between the whole numbers, if they exist (if there is no whole number in a term, the value of the whole part is taken as zero). Then the child proceeds to record the difference of the numerators and the difference of the denominators as the numerator and the denominator of the difference respectively. (Note that the child totally ignores the order of the subtrahend and the minuend when doing the subtraction process.
   Examples: $6\ ^2/_3 - 3\ ^1/_6 \rightarrow 3\ ^1/_3$; $4\ ^5/_8 - 1\ ^3/_4 \rightarrow 3\ ^2/_4$.
2. The child finds the LCM but does not enlarge the numerators but just goes to find their difference. Example: $^5/_6 - {}^1/_3 \rightarrow {}^5/_6 - {}^1/_6 \rightarrow {}^4/_6 \rightarrow {}^2/_3$.
3. The child does not cancel in the final answer. Example: $^5/_8 - {}^1/_8 \rightarrow {}^4/_8$ (should reduce $\rightarrow {}^1/_2$).
4. The child does as described in (1) but does not ignore the order of the minuend and the subtrahend. Example: $6\ ^1/_3 - 3\ ^1/_6 \rightarrow 3\ ^1/_{-3}$.
5. When borrowing, the child adds ten to the numerator as though he/she were doing integer multi-column subtraction. Note that this can still result in the correct answer if the denominator is ten.
   Example: $3\ ^1/_9 - {}^2/_5 \rightarrow 2\ ^{11}/_9 - {}^2/_5 \rightarrow 2\ ^9/_4$.

### 4.5.2 General Explanations for Observed Bugs

"There is no way of knowing exactly what the respondents were thinking, it is interesting to speculate on the incorrect strategies used" (Carpenter *et al.*, 1976).

When speculating, only explanations which seem most reasonable are presented. No doubt, other explanations could be given which can not be ruled out. However, it was deemed preferable to provide more general explanations that have applicability to other mathematical domains.

Some of the errors observed resulted due to deficient mastery of prerequisite skills, facts and concepts. Lumb (1974) and Radatz (1979) also support this view. Deficits in prerequisite knowledge include ignorance of algorithms, inadequate mastery of base facts, insufficient knowledge of the necessary concepts and symbols and incorrect procedures in applying mathematical techniques.

However, most of the errors both in the addition and the subtraction tests seemed to stem from the child 'misremembering' a rule of operation. Most of the answers obtained seem to point to the view that the children did not attach any meaning to the operations they were performing. Giving the answer $2\ ^9/_4$, which some students did, clearly supports this claim. This also confirms Hasemann's (1981) findings. Children apparently prefer to use a rule when doing a fraction problem; sometimes the rule a child uses is appropriate to the situation and sometimes not. Even in some of the responses where most of the answers were correct, the children's understanding still seems very 'instrumental' after analysing the few they got wrong; the strong impression was that they were applying an appropriate rule to the solution of a problem without knowing why the rule worked. What was needed, on the other hand, was 'relational' understanding, which is the ability to deduce specific rules or procedures from more general mathematical relationships (Hasemann, 1981). With such an understanding, the child would handle novel situations, reconsider and explain the validity of rules, reason about the domain from first principles, and use mental simulation techniques such as envisioning rather than relying solely on rules. In summary, these analyses of children's performance in the domain of addition and subtraction of fractions suggests that they view computations as purely syntactic, symbol-manipulating procedures devoid of any semantics. Vanlehn (1988a) supports this view when he observes that most students treat arithmetic procedures as arbitrary formal manipulations (i.e. "symbol pushing"). Resnick (1982) also notes that from a methodological viewpoint, arithmetic procedures are virtually meaningless to most students. It is therefore possible that it is this absence of understanding of semantics that leads to the mechanical usage of the 'buggy' procedures children so often have (O'Shea *et al.*, 1988). Nevertheless, there also seem to be other genuine reasons as to why children resort to applying rules 'blindly' when dealing with fractions. They are:

1. Educational methods influence pupils learning to solve fraction problems, and because problems in arithmetic methods mirror educational theories, then the educational methods themselves are surely suspect. Borghouts van Erp notes that as arithmetic problems become more difficult, the more abstract they become (quoted in Van Beek *et al.*, 1987). The more abstract problems cannot be solved until more concrete problems can be managed. Transposed to fractions, Van Beek *et al.* argue that the expected hierarchy in teaching fractions will be the following: pictorial -> verbal -> numerical. Bruner's theory of mathematics learning emphasises just this - that learning occurs through an enactive, iconic and finally symbolic process (Thornton *et al.*, 1983). He maintains that the student has to concretise (enact) the problem, then recall in mental image (iconic) and finally speak in words (symbolic). It appears, from our experience and also from the literature (Hasemann, 1981), that this procedure of instruction is often not followed. In fact, Martinak *et al.* (1987) found out that expert teachers rarely stressed conceptual understanding and that remediation was often procedure-oriented. As a consequence, children tend to resort to using rules without thinking about when they are appropriate.
2. The written form is comparatively complicated and, thus, difficult to understand. As argued above, if the subject is not carefully introduced, children

will tend just to treat fractions as symbols on which to apply certain mechanical rules they have learnt.

3. It is not easy to put fractions in order of size on a number line. This also ties with the two afore-mentioned reasons as it concerns representation and its relationship with mental representation. Modes of representation and their resulting mental representation have been of great interest to cognitive psychologists (Resnick & Ford, 1981). When people solve problems, they normally work with a mental representation which is an encoding of the presentation of the problem. (Van Beek *et al.*, 1987). With integers, most children tend to use the number line as their mental representation. It appears that children do not acquire a similarly good mental representation of fractions.
4. For the arithmetic of fractions, there exist too many rules and these rules are more complicated. Knowledge of many prerequisite skills is also required. There are also a multitude of different conceptual interpretations of fractions (e.g. as a fractional measure, as a ratio subconstruct or as a quotient subconstruct) which presents a major obstacle to learning to understand fraction arithmetic (Ohlsson, 1987). In effect, fractions is a more complex domain than some of the other domains, e.g. integer subtraction, decimal addition, that have been researched.
5. Fractions are used less often in daily life and are less easily described than natural numbers. "There is little doubt that the trend is away from fractions to decimals on account of metrication and the availability of calculators... Across employment we have found only limited manipulation of fractions and nothing corresponding to the more involved examples still found in school tests, employer's tests and college examinations..." (Cambridge Institute of Education, 1985).

Due to the above reasons, the semantics of fractions are not always understood by children. Thus, Ohlsson stresses that a necessary prerequisite for the design of theory-based instruction in fraction arithmetic is the construction of a semantic theory of fractions concepts (Ohlsson, 1987). It therefore seems inevitable that children resort to the mechanical usage of rules when introduced to fractions, especially if extra emphasis is not made on imparting meaning through a more careful introduction to the subject. This seems to be generally true of many arithmetical domains, as it appears that a very large part of the population apply rules in general without understanding their significance (Dalenoort, 1988). This could be easily ascertained as arithmetic provides a good test bed; its rules are clear, and the nature of mistakes are, in principle, traceable. This observation is essentially simple but important, and so is expressed in the following principle:

**The Principle of Relational Arithmetical Understanding.** In order to be able to consistently and appropriately apply the correct rules in arithmetic, real or conceptual understanding is an important and necessary prerequisite, without which arithmetic will be viewed as a purely syntactic, symbol-manipulating exercise devoid of any semantics.

### 4.5.3 Explanations for Observed Bugs using Bug Theories

Some more general principles and theories were considered to provide explanations for the observed bugs. One possible general explanation is that when a child does not know the procedure for solving a particular problem, he/she constructs one by analogy with a previously learnt procedure for solving a different problem. Since, as argued above, children ignore the semantics of fractions, there is a clear analogy between the *columns* of decimal representation and the *rows* of fraction representation. After all, to add 13 and 12 they know that they add the *columns* separately. Therefore, their line of reasoning may be that, to add $^3/_4 + ^1/_5$, why not add the *rows* separately, especially when they do not know what to do or if they have not been introduced to (or have forgotten) the process of fraction addition. Besides, "it is natural because in many cases we do add things that belong to the same category. We add apples to apples, chairs to chairs, tens to tens, and hundreds to hundreds" (Hershkowitz *et al.*, 1980). The same argument applies for subtraction problem, e.g. $^5/_7 - {}^3/_4 \rightarrow {}^2/_3$.

Several of the current theories were able to explain some of the bugs observed in this study. Brown & Vanlehn's (1980) Repair Theory seemed to provide very neat explanations to some. For example, take the situation where a child was given the problem $2\ ^1/_6 - 1\ ^2/_3$. The child correctly proceeded to the stage $1\ ^{(1-4)}/_6$, after having found the LCM, enlarged the numerators and subtracted the whole numbers. At this stage, the child appeared to arrive at an impasse, probably because he did not remember the 'borrowing' rule (since $1<4$) or he had forgotten. He proceeded to repair the step doing, $1\ ^{(4-1)}/_6$ and went on to obtain the erroneous answer: $1\ ^3/_6 \rightarrow 1\ ^1/_2$.

Matz's (1982) theory could also explain some of the bugs but, probably, not as neatly as the Repair Theory. All the students tested had done fraction multiplication which basically uses the rule $^a/_b \times {}^c/_d = {}^{(a\times c)}/_{(b\times d)}$. The child, one could argue, could have totally forgotten the whole addition process since it is more complicated (it requires a lot more rules) but vividly remembers the multiplication rule. Hence, he/she uses the multiplication rule (*as it is*) to solve the addition problem. For example, take the question where some children produced $^1/_4 + {}^1/_6 \rightarrow {}^1/_{24}$. However, most of the children actually produced $^1/_4 + {}^1/_6 \rightarrow {}^2/_{10}$. By the same argument, in the second case, the existing multiplication rule was *adapted* to solve the new problem (i.e. addition problem) giving: $^a/_b + {}^c/_d \rightarrow {}^{(a+c)}/_{(b+d)}$. This resulted in the solution $^{(1+1)}/_{(4+6)} \rightarrow {}^2/_{10}$.

Young & O'Shea (1981) found that they could account for a large proportion of the observed errors in children's subtraction, by the deletion of one or more rules or the inclusion of some mal-rules in the child's production system (PS). This theory also neatly explains some of the bugs but not others. For example, one could mimic a child not cancelling in the final answer by deleting the 'cancelling' rule from the child's production system. So a child would provide an answer as $^2/_{10}$ instead of further cancelling to give $^1/_5$. Psychologically, this could be explained as the child having forgotten the cancelling rule or never having been introduced to it in the first place.

### 4.5.4 What's the Point?

Theories which purport to classify or explain whole classes of bugs must ultimately be judged on their usefulness. An understanding of the genesis of bugs has crucial implications for remedial actions, as well as for pedagogical strategies that endeavour to avoid the generation of these bugs in the first place. Of course, these remedial actions and pedagogic strategies are key issues of immense concern to the ITS community. In the fractions domain, the evidence discussed above shows that several of the current theories could provide convincing explanations for some of the observed bugs, but none were able to explain all of them, whereas some bugs could be explained by more than one theory, albeit with varying degrees of success. Since there are also likely to be domain-specific features (e.g. Sleeman (1984a, 1984b) argues that 'parsing' mal-rules stem from severe misconceptions about the particular subject domain and the notation used, or about prerequisite knowledge), it must be accepted that, for the present at least, no unified theory of bug causation is (or may ever be) available. It is further complicated by noting that other factors such as learning environment, interest level of the student, age and external pressures will probably need to be taken into consideration. In effect, what seems to be required is a broader, possibly composite, generative theory of bugs (Wenger, 1987, page 198). The conclusion then seems to be that intelligent tutoring systems will have to be able to take account of multiple theories.

The purpose of having theories of bug causation is to use them to modify teaching strategies and thus devise learning environments which prevent the bugs from occurring in the first place. It would appear that the present state of research makes this extremely difficult, for two reasons namely:

1. It seems as if a multiplicity of theories need to be taken into account, and since more than one may be able to explain a particular bug, its prevention may apparently require several independent alterations in the teaching strategy.
2. Perhaps more fundamentally, current theories of bug causation are at too low a level. For example, the Repair Theory of Brown & Burton explains why a particular bug arises (because the child arrives at an impasse when a procedure has been incorrectly or wrongly applied to a problem, and carries out an erroneous repair) but does not explain why children in general tend to reach these impasses in the first place. Put in another way, it does not attempt to explain or model the genesis of a student's core procedure, but instead perturbs the correct skill so that an impasse is generated. Surely, a better model of impasse generation requires a theory of how procedural skills are acquired through the kind of teaching sequences children encounter at schools (Vanlehn, as quoted in Wenger, 1987). It seems fair to say that the sort of model Vanlehn has in mind here might provide clues on how buggy core procedures are generated and therefore bypass the 'impasse generation' problem. Sleeman (1984a, 1984b) supports this view when he claims that the origin of 'parsing' mal-rules are deeply rooted in the learning process. He argues that they are so fundamental in some cases that they may not even be viewed as variants of correct rules.

Thus, Vanlehn has since stopped looking within the domain for explanations of how incorrect procedures are formed and instead turns his attention now to the teaching sequence. He has recently proposed the STEP theory as mentioned in Chapter 3. The STEP theory, by attempting to account for the *generation* of buggy core procedures, is a major step in the right direction. Ross (1987) also supports this view. The felicity conditions it proposes would certainly be of interest to the ITS community as it provides certain pedagogical principles or guidelines which could be followed when constructing ITSs, at least in arithmetical or mathematical domains.

It is clear that there is still much scope for more work. New theories of bugs, of any type, are much needed and can constitute an important contribution to the field. Further research is needed both at the empirical (or practical) level of collecting, classifying and trying to account for bugs similar to the work reported in this chapter, and also at the more theoretical level of trying to understand the fundamental cognitive processes and essential ingredients underlying knowledge acquisition and human learning in general. From a psychological perspective, ITSs furnish good test beds for learning/mislearning theories. In fact, Anderson (1984) concludes that there seems to be a very intimate relationship between issues in cognitive psychology and intelligent tutoring which goes beyond the relationship between science and its applications. Therefore, these investigations should also result in widely applicable contributions to the construction of intelligent tutoring systems.

## 4.6 Concluding Discussions and Implications for Building an ITS

Previous work, the experience gained with fractions, and the results and explanations of the empirical as well as the bug theoretical work reported and discussed above raise several questions as to how to go about building truly intelligent tutoring systems, not only for the fractions domain, but also for other 'complex' mathematical domains as well a suitably structured domains.

The mal-rule methodology for constructing an ITS discussed in the previous chapter involves extensive analysis of the protocols of students drawn from the target population in order to build up a library of bugs which can then be used in error diagnosis. However, it is clear that even very simple domains can yield a very large number of bugs - for example, Brown & Burton (1978) identified over 130 different bugs (110 primitive and 20 compound) in the subtraction domain. Worse still, a study of subtraction problems (Vanlehn, 1982) shows that 37% of the students manifest multiple bugs, sometimes as many as four together. Surely, this suggests that the search space for all the feasible bugs will be too large for exhaustive testing. Payne & Squibb (1987), after an in depth study of bugs in algebra, similarly conclude that too many mal-rules are needed to diagnose all errors and worse still, they observed that different mal-rules appear to be needed to model different populations. How much effort, time and resources should ITS designers put into this analysis phase? In this analysis of the addition and subtraction of fractions, five bugs accounted for 90% and 85% of all the errors observed respectively. Stevens *et al.* (1982) also demonstrated that sixteen bugs accounted for 58 to 72 percent of student errors in their responses to human tutor questions. Similar results are also reported in Payne & Squibb (1987). If this is generally true, then building extensive bug libraries seems inefficient, since most of their content will rarely if ever be used. Human tutors can not and do not bother to predict all the bugs children can come up with; instead they resort to strategies like reasoning through the child's solution or simply reteaching their preferred method. Gilmore & Self (1988b) suggest that the application of machine learning techniques to ITSs would be inevitable to relieve such costly domain analyses.

The pen-and-paper test results suggested that there was considerable instability in the bugs exhibited in the fractions domain. 74% and 64% stability were observed for the addition and subtraction processes respectively. (Evertsz (1982) independently came up with a figure of 79%.) While the accuracy of these figures depends on what one defines stability to mean, after all one person's "noise" is another's "interesting bug", it nevertheless raises the biggest challenge to the mal-rule approach. This is because it challenges the fundamental basis of this approach. The conventional mal-rule approach assumes that bugs represent *consistent* mal-rules; but children were observed employing two different buggy procedures in solving the same problem within a short session. In two reliability studies, Vanlehn (1982) found that only 12% of the students had similar bugs in tests given a few days apart, and even long term reliability data were similarly unstable. Therefore, classifying children's behaviour on some single occasion as due to various types of buggy behaviour may be terribly misleading. Systems based on this approach claim remediation could easily be done after the diagnosis. However, due to the problem of instability, an important precondition for remediation is invalidated (Sleeman, 1987c).

In a 'complex' domain like fractions, the problems are even worse, because fraction problems may involve some or all of the skills of integer computation. Any bugs in these underlying skills will show up as errors in fraction computations. For example, fraction addition may involve some or all of the skills of integer addition, multiplication and division, finding equivalent fractions, and finding the lowest common multiple (LCM). Fraction subtraction may involve all of the above with the inclusion of the subtraction skill itself. Ideally, a fractions bug catalogue would have to contain all the permutations of the bugs in these sub-skills, as well as those specific to fraction computation itself. As mentioned earlier, a study has already yielded 68 primitive bugs in addition of fractions. A more detailed study would surely result in a larger figure. This is a classic combinatorial problem, and the resulting number of bugs is likely to be prohibitive. However, some argue that the whole point of Artificial Intelligence is to be able to solve such problems (Dumont, 1988).

The problem of bug instability would also increase, since instability in any one of the mal-rules in the component sub-skills would show up as an apparent instability in the overall mal-rule.

Even if sufficiently extensive bug catalogues could be constructed, it is clear that *post hoc* diagnosis would be difficult if not impossible. The more steps there are in a computation, the more likely it is that very different bugs can produce the same overall result. In other words, mal-rules are intrinsically ambiguous. Some systems (e.g. DEBUGGY (Burton, 1982)) resort to generating other problems which could distinguish amongst these bugs so as to isolate the error. This would be a very time-consuming process in a complex problem, and of doubtful pedagogical value: human tutors would not normally set 10 extra problems simply to diagnose a single mal-rule; rather, they would work through the child's solution. In addition, there is still a high percentage of unexplained errors (~30-40%) (O'Shea *et al.*, 1988). For example, Vanlehn (1982) reports that 34% of the errors made by students in several samples could not be diagnosed by DEBUGGY. This is not a particularly convincing 'hit rate'.

The problem of diagnosis can be illustrated by considering an example from the testing session. After the main test, a student was given the extra problem $^7/_{20} + ^2/_5$ while trying to diagnose his misconceptions. He found the LCM as 20, and then found the equivalent fractions correctly as $^7/_{20}$ and $^8/_{20}$, but added up 8 + 7 to give 5 i.e. he failed to 'carry over' when adding, or he may simply have made a 'slip'. (Slips are unsystematic, "careless" errors: e.g. fact errors such as 7-3=5 (Vanlehn, 1988a)). To make matters worse, he possessed the cancelling skill so that $^5/_{20}$ rightly became $^1/_4$. Given only an answer of $^1/_4$ to $^7/_{20} + ^2/_5$, without access to intermediate step(s), could any useful diagnosis be done? It is likely that such an error would appear purely random.

Even when 'diagnosis' is possible, it is of limited value. For example, Attisha & Yazdani (1984) describe a system that is able to diagnose students' multiplication errors. When a student works a two digit multiplication problem incorrectly, the system responds as follows:

$$\begin{array}{r} 87 \\ \times\ 43 \\ \hline 32 \end{array}$$

Your answer is wrong.

Possible causes of error:

1. You multiplied the number in the multiplicand by the number directly beneath it in the multiplier, and you wrote down the carried number, ignoring the units number.

The system clearly displays its complex diagnostic capabilities. But Lepper & Chabay (1988) question how children would feel or react to this information. They also question whether they would be able to make use of it or if they would want to try again. In this study, in response to the problem $^5/_7 + ^4/_7$, most of the children produced $^9/_{14}$. Diagnosing the bug as "the child adds both numerators and both denominators to obtain the numerator and denominator to the answer respectively" achieves little. The child clearly does not possess the *overall* fraction addition procedure, but he/she may or may not have the skills of finding the LCM of two numbers or the other skills that are prerequisites for solving this problem. Only if it can be fully demonstrated that the child possesses these skills can it be concluded that he/she lacks only the skill of understanding how and when to 'merge' them together.

The credibility of the 'traditional' mal-rule methodology is thus seriously questioned in more complex domains such as fractions, at least as embodied in systems such as LMS/PIXIE (Sleeman, 1982b, 1985a), BUGGY (Brown & Burton, 1978) and DEBUGGY (Burton, 1982) which consider only the final answer in their diagnosis. Boden (1983) has described the basic assumption that underlies all three as being that a bug is an "optimistic rather than defeatist notion. For it implies that elements of the correct procedure or skill are already possessed by the student and that what is wrong is a precisely definable error that can be identified and fixed". In more complex domains, there are just too many 'precisely definable' errors, so that the bug catalogue is prohibitively large and the *post hoc* diagnostic process too difficult.

The solution seems to be to ensure that intermediate steps are available when attempting diagnosis in a more complex domain. In fact, Payne & Squibb (1987) put it more strongly: "This demonstrates that students' intermediate steps towards a solution are not merely a convenience for cognitive diagnosis: they are necessary in principle". Diagnosis requires the identification of those skills that are possessed by the child, as well as those that are not (Ashlock, 1976). As is well illustrated by the examples above, this cannot be achieved in a complex domain without a model-tracing methodology which will be able to understand the student's behaviour as it is generated (Anderson *et al.*, 1985b). As the student works through a problem, the tutor should be able to determine the rule or mal-rule that led to that step being taken, and should be able to interrupt with advice as soon as an error occurs. Lewis *et al.* (1987) and Visetti & Dague (1987) have also had some success in building systems incorporating some model-tracing techniques.

A conclusion also arrived at is that current theories of bug causation are at too low a level to make much impact on the construction of ITSs as they should do. They would (if they exist) have great implications for remedial actions as well as modifying pedagogic strategies and generally devising learning environments which pre-empt these bugs from occurring in the first place. These constitute some of the characteristics of the environments that those in the ITS community are striving for. If current theories prevail, then a multiplicity of theories might need to be taken into account when constructing ITSs; this looks very ominous, at least, at this present stage of research. Their main drawback, exposed by the bug theoretical study, is the fact that they appear to take a very narrow and superficial view of how bugs arise in students' computations. Most of the bugs observed in the children's protocols seem to stem from a distinct fundamental lack of *understanding* of the required fraction concepts. Hence, mal-rules' genesis appear to be more deep rooted in the learning process than is perhaps presently acknowledged. In this respect, Vanlehn's STEP theory, though it has its limitations, is a major step in the right direction as it 'views' children's bugs at a much 'deeper' level than the other theories do; it also provides some clues (felicity conditions) to abide by when constructing ITSs in the domain of mathematics. These guidelines were instrumental in the development of the tutor reported later on in this thesis.

## 4.7 Summary

This chapter has reported on some pilot studies that were carried out in order to provide a better basis to address the central goals of this research, than would have been with only a review of the literature.

The next chapter, drawing from this as well as previous chapters, concretely specifies what the two main issues addressed in this research are, and goes on to address the first of them.

# Chapter 5

# Problem and System Specification

## 5.1 Introduction to the Problem

The review of the literature in Chapters 2 and 3 outlined the research that has been carried out in the area of Intelligent Tutoring Systems (ITS), with particular emphasis on mathematical domains. Some of the limitations of these systems were also highlighted. However, although many tutoring systems have been constructed for the domain, there still exists no consensus on any one approach for the design of such systems. Chapter 2 reports on a reasonably successful intelligent tutoring system, GUIDON (Clancey, 1983, 1984, 1987), which was developed using the approach of adapting a pre-existing expert system. Clancey and his colleagues constructed GUIDON by adapting the famous MYCIN expert system (Shortliffe, 1976). Nonetheless, such an approach is more geared towards discourse domains such as medicine and geography and is not of much value in such structured domains as mathematics. In any case, some researchers have already echoed rejection of such an expert system-based approach to constructing ITSs (e.g. Self, 1985a; Todd, 1988). The long-term goal of ITS experts is, surely, to produce tools and techniques for constructing ITSs so that other experts could use them for building ITSs for their domains of expertise. However, Clancey had to carry out an extensive review of the medical literature as well as meet regularly with a physician consultant with the purpose of revising MYCIN's rules to make the teaching points become clear (Clancey, 1983, 1984, 1987). Arguably, it would have been better for the expert physician consultant to construct GUIDON himself/herself if the required tools and techniques were available. Therefore one of the important legacies ITS experts should leave behind in the long-term is a credible approach for constructing these systems. So far, attempts in this vein are either nonexistent or inadequate.

There seem to be two main reasons for the lack of success in establishing a credible approach for constructing ITSs as a whole in the domain of mathematics.

1. Cognitive scientists have failed to come up with a universally acceptable theory of instruction and learning (Wenger, 1987). Most current theories of instruction and learning are woefully inadequate in their ability to have any impact on the construction of ITSs. "What is needed is a stronger theoretical foundation based on a deeper analysis of acquisition of complex learning-to-learn, conceptual and performance knowledge" (Mandl & Lesgold, 1988). This is one of the reasons why the development of ITSs, not only for mathematical domains but for all other domains, is still in its infancy. However, this makes the area increasingly vibrant. For example, many have come into the area to test out their various theories (e.g. John Anderson).
2. There has been little research into providing such a credible approach (or approaches) (possibly because researchers are waiting on a more universal theory of instruction and learning). Only a handful of researchers have attempted to provide general frameworks for ITS construction (e.g. Davies *et al.*, 1985; Ikeda *et al.*, 1988; Hartley & Sleeman, 1973; O'Shea *et al.*, 1984). Most ITS designers have actually focussed on various components of the ITS architecture, such as the student model, the domain expert or the user interface, as stand-alone parts rather than on the effectiveness of the system as a whole (Barzilay, 1985; Ford, 1987a, 1987b, 1988). For example, Brown & Burton (1978) in BUGGY and Sleeman & Smith (1981) in LMS focussed on student modelling. This is by no means to belittle the work done on the various components of the ITS but merely to say that there is sufficient evidence in the literature to suggest that it is not trivial to incorporate these resulting stand-alone components into fully fledged ITSs. For instance, Sleeman (1987c) reports that even though PIXIE (a Lisp-based ITS for algebra) "has a model for student's problem solving, it has not so far proved possible to remediate very

effectively". Nevertheless, work done on these components has been invaluable in some of the more successful existing ITSs and will surely continue to be so.

Realistically, the solution to the first of these two problems is still well away, so that it would probably be better for researchers to concentrate more on the second problem in the short-term, with more experimentation on various approaches. Perhaps some of the energies being spent by researchers working on various components of the ITS should be redirected towards this goal. In effect, ITS experts should probably adopt more of the 'engineering approach' rather than the more rigorous scientific style. Wenger (1987) notes that, as a developing field, ITS needs more laboratory work. He argues that the pragmatic needs that motivate ITS research are very concrete and immediate, and justify such laboratory experiments. Nevertheless, the crucial work being done towards the provision of a more universal theory of instruction and learning should by no means be neglected.

As discussed earlier on in Chapter 3, current approaches to the construction of ITSs in mathematics can be grouped under two main headings:

1. The mal-rule approach
2. The model-tracing approach

It was also noted that most current systems are based on the mal-rule approach. In Chapter 3, the steps involved in the mal-rule approach were listed, along with some of the main criticisms levelled at it, using DEBUGGY as a typical example. More importantly still, using fractions as a test domain in Chapter 4, the inadequacies of this approach as a credible method for constructing ITSs were clearly highlighted. FRACTIONLAND also further clearly demonstrated this approach's limitations. As a consequence, it was argued in the latter part of Chapter 4 that a model-tracing type approach is preferable to the mal-rule one, at least for realistic domains.

The previous paragraph suggests a model-tracing type approach should be adopted. The steps involved in this approach were highlighted in Chapter 3. However, this approach also suffers considerable problems, at least as exemplified by current existing tutors based on this approach. Some of these shortcomings were also discussed in Chapter 3.

In summary, the two predominant approaches to developing ITSs in mathematics have been mentioned and their shortcomings have been highlighted. These shortcomings of both approaches have contributed to a lack of consensus on an approach for the design of such systems. It was not the intention to provide a comprehensive set of all criticisms levelled at ITSs based on these two approaches. In fact, some might even consider these criticisms to be the 'tip of the iceberg'. All is clearly not well with ITSs, and although answers to all of these problems are not expected in the immediate future, it is hoped future attempts at building ITSs will tackle at least some of these shortcomings.

## 5.2 Aims of Research (Problem Specification)

The objectives of this research are hence twofold:

1. To investigate an alternative approach for building ITSs in mathematics, which supports an iterative style, and which improves on at least some of the shortcomings of the two existing ones.
2. To develop and test an experimental system which is based on this approach for the domain of the addition of fractions.

## 5.3 The Iterative-Style Approach (System Specification)

Analysis of existing approaches, the experience gained with the reasonably complex domain of fractions, the review of the literature in Chapters 2 and 3, conclusions drawn from the empirical as well as bug theoretical work reported in Chapter 4, and previous work of other researchers have all contributed to addressing the first of these aims. They not only provide ideas on how to approach building an ITS for the fractions domain, but also for other 'complex' mathematical domains as well as suitably structured ones. This iterative-style approach (essentially a much-improved model-tracing approach), like the other two, consists of a list of features (along with some brief justifications) which an ITS should possess. In brief, the experimental system to be developed for the domain of addition of fractions should demonstrate the following features; the key **novel** ones being the abilities to:

1. *Pre-model the student.* The bug theoretical study reported in Chapter 4 highlighted the fact that some errors children make result from deficient knowledge. "The major problems in automated tutoring is that the system needs to know what the student knows and understands, what his misconceptions are" (Barzilay, 1985). Pre-modelling should identify these prerequisite skills, concepts and facts that are (and are not) possessed by the student in question as well as his/her coverage of the syllabus. If the student does not exhibit enough prerequisite knowledge, then necessary prerequisites should be presented (Woolf *et al.*, 1988). Chapters 2, 3 and 4 emphasise the fact that to individualise instruction, the system should have some model of the student. Besides, it also seems unreasonable to assume that all learners start off with the same structuring of space (Self, 1987b). O'Shea's (1982a) tutor makes does some form of pre-modelling but only makes use of the pre-test score.
2. *Preteach prerequisite skills.* Most of mathematics is considered hierarchical in nature, that is certain skills are prerequisites to higher level skills (Oliver, 1973). As discussed earlier, the empirical study reported in Chapter 4 clearly shows that errors resulted due to deficient mastery of prerequisite skills, facts and concepts. It is inevitable that if students do not possess the prerequisite skills, they would encounter difficulties with the goal topic. For

instance, Moore & Sleeman (1987) point out that if a student consistently solves tasks such as $2 * 3x + 4x = 22$ as $x = {}^{22}/_{14}$, then tutors should recognise this consistency in making precedence errors and probably switch focus from algebra to arithmetic precedence until the student shows signs of understanding precedence in arithmetic, and then resume algebra tutoring. Most importantly, this will also strengthen links with other areas of mathematics. Ridgway (1988) rightly points out that current systems (e.g. DEBUGGY/Geometry Tutor) support a split between algorithm and applications as they view topics they tutor as separate from 'mathematical thinking' (see Chapter 3). The ITS should then be able to teach (reteach) the necessary prerequisites prior to teaching the goal skills. Another likely advantage is that preteaching would also make the diagnostics or any information generated by the tutor more meaningful, as the student would better understand what he/she is expected to do.

3. *Provide examples (sequence of examples/lessons) such that Vanlehn's felicity conditions are satisfied.* The bug theoretical study highlighted the fact that, so often, teachers or books assume too much of the students in their examples as well as their lessons. Ross (1987) supports this view. Vanlehn's (1987) felicity conditions address this issue.

4. *Keep information about the student's ability and progress (model the user).* The review of the literature noted that student information databases, such as student models or student history files, are conspicuously absent from model-tracing tutors, at least as exemplified in the Geometry and Lisp tutors (Anderson *et al.*, 1985a). However most researchers seem to agree that to provide truly individualised instruction, which ITSs strive (or should strive) to do, such information is vital (Gilmore & Self, 1988; Hartley & Sleeman, 1973; Rich, 1979; Ross *et al.*, 1987; Self, 1974, 1987b, 1988b, 1988c;

Sleeman, 1985c; Tobias, 1985; Wachsmuth, 1988; Zissos & Witten, 1985). In brief, the student model is an indispensable component of an ITS.

5. *Support various idiosyncratic ways which the student might choose to solve the problem.* This is contrasted with the ideal problem solver of the Lisp and Geometry tutors which are 'hardwired' with the 'correct' way of solving the problem (see Chapter 3), or with mal-rule tutors, which only normally request the final answer, as discussed in Chapter 4 and demonstrated by FRACTIONLAND. An ITS should be capable of comparing solutions and indicating to the student if his/her correct solution is non-optimal, possibly then present a more optimal solution. It must also be capable of solving the problems it presents to the students (Woolf, 1987).

6. *Explicitly represent knowledge, e.g. of the student, of tutoring strategies, of the topic to be taught, of the curriculum and of the more enduring characteristics to which instruction should be sensitive.* The case for this was also argued in some depth in Chapters 2 and 3 that traditional CAI's main weaknesses stem from its implicit representation of knowledge. Current mal-rule and model-tracing ITSs have also been noted in previous chapters to suffer from this limitation as well. It was argued that a system would be more 'intelligent' if this knowledge were all made explicit. Woolf (1987) dwells on this premise.

7. *Test the student's understanding.* This might be the student's understanding of prerequisite skills (e.g. as in feature 1) or his/her understanding of the goal skill. The results of the testing exercise should be used to modify the student model.

8. *Motivate and support a more flexible style of tutoring.* Both the review and the pilot study point out that factors such as boredom and motivation are involved in learning (Duchastel, 1986). Suppes (1988) notes that the absence of sustained work on motivation in ITS research is its most serious

omission. However, it is still extremely difficult to detect or model such phenomenological factors. As a safeguard, ITSs should be as motivating as possible to reduce the possibility of the student getting bored. 'Buggy' tutors were demonstrated to be very boring and uninteresting in their style of tutoring through FRACTIONLAND. Model-tracing tutors were also noted to be inflexible (see Chapter 3). ITSs should be more flexible, e.g. by playing a more collaborative/mentor role for the weaker/better students respectively (Barzilay, 1985).

9. *Provide environments in which the interaction between them and students should be as close as possible to the reality of the day.* In this era, where traditional classroom instruction is still the norm and pencil-and-paper still remains the dominant technology, ITSs developed should reflect these realities if they are to be truly useful. In Chapter 3, it was pointed out that children experience the 'training wheels' problem when they leave sophisticated environments such as ALGEBRALAND and have, e.g. to sit our present examinations using pen and paper. Nevertheless, one must not become so conservative as not to realise that the computer revolution will continue to have fundamental repercussions on educational practices.

10. *Present a transparent view of the approach on which it is based.* ITSs should be easy to understand and the approach on which it is based should be transparent so that it could be applied in other suitable domains. It would also make subsequent system enhancements/modifications much easier.

In addition to the above features, the prototype ITS to be developed should also demonstrate, perhaps even better, abilities which current model-tracing tutors (e.g. the GEOMETRY tutor) do namely:

11. *Monitor the student step by step.* The case for monitoring (or model-tracing) has been clearly put in the previous chapter. Farell (1987) enforces this view

as he notes that " ...the ideal educational system will carefully monitor interaction with the student ...". Besides, Chapter 3 also noted that some systems (e.g. PIXIE) which are based on the mal-rule approach have been enhanced to make use of intermediate steps, albeit in a very restricted fashion, in order to discriminate amongst possible models, which is a step towards a more monitoring-type approach. In summary, the system should then be able to monitor and comment upon the student's actions, recognise optimal, less than optimal, and clearly irrelevant actions. It should also offer help, hints, explanations and tutoring advice as appropriate (Woolf, 1988).

12. *Diagnose in a problem-solving context.* Diagnosis becomes easier and more efficient as the system has access to intermediate steps. This is because the problems of uncertainty and/or combinatorial explosion faced by DEBUGGY and PIXIE when carrying out diagnoses (assigning credit/blame) are substantially reduced (see Chapters 3 and 4). This point is well expressed in Self (1987b): "an ICAI system should endeavour to interpret students' inputs, not merely in terms of knowledge understood or not, but as evidence about the student's goals, i.e. what he is trying to achieve".

13. *Enable the student to communicate his/her intentions (plans) prior to executing them.* This follows from the previous feature: diagnosis will be even further enhanced if this is done; for example, if the student specifies what operation he/she intends to carry out rather than the system trying to infer it. Self (1988b) also stresses that ITSs should avoid guessing and should get the students to tell them what they need to know.

14. *Remediate in a problem-solving context.* As discussed in the pilot study, systems of the 'buggy' type print the inferred bug. This has been shown to be of limited value as there is no point diagnosing if one cannot remediate (Self, 1988b). Remediation would also be easier and of more use to the

student if the system is monitoring every intermediate step. It should take the form of diagnosing the error, providing the correct answer, instructing the student what to do next or giving hints (Lepper & Chabay, 1985). Questions of where and when to interrupt, what feedback and/or which style of remediation is most appropriate to provide are still unresolved issues which researchers are currently addressing (e.g. Woolf & McDonald, 1984). Current model-tracing tutors achieve this feature to a certain degree; however, this approach strives to support more improved remedial strategies than current ones.

15. *Maintain control over the whole tutoring endeavour and support a more mixed-initiative interaction.* They should be simple to use, able to detect and report minor/syntactic errors and maintain a good level of interaction between it and the student. However, it should also support a more mixed-initiative interaction. FRACTIONLAND demonstrated that 'buggy' tutors mostly limit students to monosyllabic answers. Even current model-tracing tutors do not escape this criticism. Giving the child more initiative, and getting him/her to interact more with the tutor is also likely to reduce the possibility of boredom.

Many of the novel features of the approach are corroborated in Woolf *et al.*, (1988), where they present a system, called TUPITS (an acronym for TUtorial discourse Primitives for Intelligent Tutoring Systems), which is an object-oriented framework for defining primitive components of a tutorial discourse interaction. TUPITS has been used to build the Recovery Boiler Tutor, a language tutor and a physics tutor (Woolf & Cunningham, 1987). They intend to evolve the framework until it reaches the stage where it could be used within an authoring system to reduce the excessive time required for constructing ITSs. Their goal is therefore to provide tools and techniques for ITS construction. In this respect, their work is similar to that reported in this chapter.

## 5.4 Summary

So far, this chapter has addressed the first of the two aims of this research (see Section 5.2). Putting it more succinctly, the *novel* features of the proposed 'iterative-style' approach over current mal-rule and model-tracing ones include:

- Pre-modelling of the student.
- Preteaching of prerequisite skills.
- Providing tutoring sequences such that Vanlehn's felicity conditions are satisfied.
- Modelling of the student throughout all his/her interactions with the tutor.
- Explicitly representing knowledge, e.g. of the student or of tutoring strategies.
- Supporting various idiosyncratic ways which the student might choose to solve the problem.
- Testing the student's understanding.
- Supporting improved remedial strategies.
- Motivating and supporting a more flexible style of tutoring.
- Providing environments in which the interaction between them and the students is as close as possible to the reality of the day.

The major part of the rest of this thesis addresses the second aim - to develop and test the a system based on the iterative-style approach for the domain of addition of fractions. (Due to time constraints of this project, it was decided not to include fraction subtraction in the system, even though it was empirically researched. Besides, the suggested approach should be as applicable to it as to other complex mathematical domains.)

# Chapter 6

# System Overview

## 6.1 Introduction to the System Overview

In order to improve on at least some of the shortcomings in existing mathematical ITSs, a new approach to their construction has been proposed - an improved model-tracing approach. This iterative-style approach, like other approaches, consists of various principles; fifteen in all were presented in Section 5.3. This chapter starts to address the second of the two central issues of this research, i.e. the development and testing of an experimental system based on the iterative-style approach for the domain of fraction addition (see Section 5.2).

FITS is the acronym for the system developed, standing for **F**ractions **I**ntelligent **T**utoring **S**ystem. It was designed to have a clear separation between domain-independent and domain-dependent parts, in order to facilitate the incorporation of a new domain. The system also has a clear and simple architecture. Conceptually, each student-tutor interaction cycle involves the flow of data through a number of components. All these components are, as far as is practical, rule-based to permit ease of subsequent enhancement. Furthermore, such rules also lend themselves easily to automatic manipulation (O'Shea, 1979).

## 6.2 Language and Environment

FITS is a Quintus Prolog-based system; Quintus Prolog also happens to be one of the most powerful implementations of Prolog. The reasons for using Prolog have already been provided in Section 1.4.

FITS was developed on a SUN-360 workstation running under UNIX. It consumes 380Kbytes of memory but requires 680Kbytes when the necessary library predicates are imported. It takes about 3 minutes of processor time to be compiled.

## 6.3 Overview of FITS's Architecture

Figure 6.1 provides a top-level description of the architecture and flow of control in FITS.

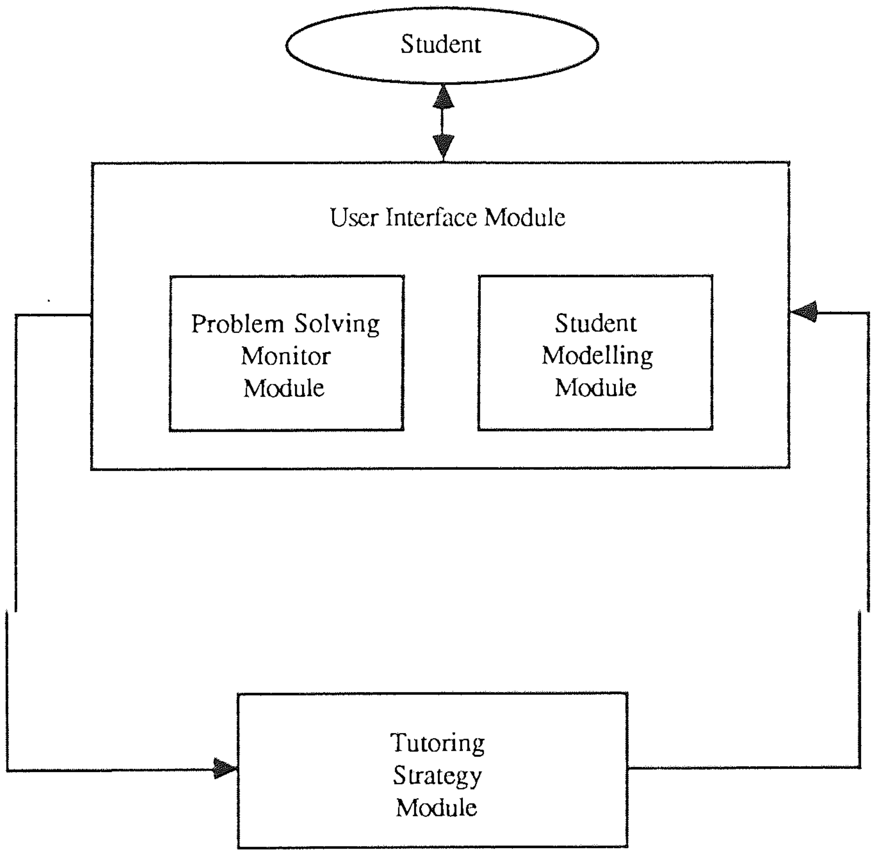


Figure 6.1 - Simplified Architecture of FITS

Every input from the student is processed by the User Interface module or Administrator. This User Interface module actually comprises the Problem Solving Monitor module and the Student Modelling module. There is no separate User Interface module as such; rather, these two modules between them perform the function of a typical User Interface module (see Section 2.4.1) as well as their own separate functions. The User Interface module is presented for explicatory purposes. Self (1988a) explains this quite succinctly: "The division of an ITS into the standard 'subject, student, tutor' module is an explicatory device, not a guideline for ITS implementation".

After processing input from the student, the User Interface module invokes the separate Tutoring Strategy module. This component in turn passes control back to the User Interface module, which generates an output message for the student. Although the roles of typical ITS components were discussed earlier on in Chapter 2, the functionality of these components in FITS is next described.

### 6.3.1 Active Modules

The *User Interface Module's* role is to control the interaction between the student and FITS. The student interacts with FITS in terms of integers (e.g. a menu option (1, 2, ...)) or by fraction addition expressions (e.g. $2{:}^1/_4 + 3{:}^1/_2$, $2{:}^1/_3$, etc). The User Interface module therefore accepts such inputs from a student and checks them as far as it can. This involves checking the syntax of the input and reporting various minor/syntactic errors. This module provides various menus for the student to choose from, presents FITS's output to the student, etc. At the same time, it also attempts to maintain a reasonable level of dialogue with the student to limit his/her possibility of boredom. However, as explained earlier, the User Interface module comprises two components: the Problem Solving Monitor and the Student Modelling modules.

The *Problem Solving Monitor Module* as its name implies, is the monitoring (model-tracing) component of FITS; hence it is a central module to the system's operation. Basically, it is the expert system of FITS but with much more. It has many features which include the following:

1. It generates fraction addition problems for a given particular level of difficulty to present to a particular student.
2. It is capable of solving all the problems it generates.
3. It monitors the student and comments sensibly on his/her attempts.
4. It allows for various idiosyncratic ways of solving the given problems.

5. It allows for student generated problems to be presented to the system.
6. It provides hints, examples, help, etc., i.e. it provides remediation depending on what teaching strategy is being carried out at the time.
7. It diagnoses 'simple' misconceptions in the student's understanding.
8. It quizzes the student to test his/her understanding of a particular concept or task.
9. It presents expositions of various concepts/tasks.
10. It gets the student to communicate his/her intentions prior to executing them.
11. It compares the student's solution to that of the system's so as to enable the student to learn more optimal ways of solving the problem, especially if his/her approach is very long-winded.

The *Student Modelling Module* dynamically maintains a student model. It attempts to keep an up-to-date representation of what it believes to be the student's emerging knowledge of the skills and concepts of fraction addition. As would be imagined, it has to work in very close collaboration with the Problem Solving Monitor module. It has two main functions:

1. When a student logs on the system for the first time, it uses his/her answers to a pre-test to initialise his/her student model (pre-modelling).
2. It subsequently up-keeps this model by maintaining a student-tutor history file. This will be explained later on in the thesis.

The *Tutoring Strategy Module* is the component that decides the next teaching action. This involves selecting a new and appropriate concept to teach, deciding whether to present (re-present) a concept or to provide a hint or an example, etc. The User Interface module always readily invokes the Tutoring Strategy module. When the latter decides on what to do, it immediately instructs (i.e. passes control back to) the User Interface module to execute its decision. For example, imagine the

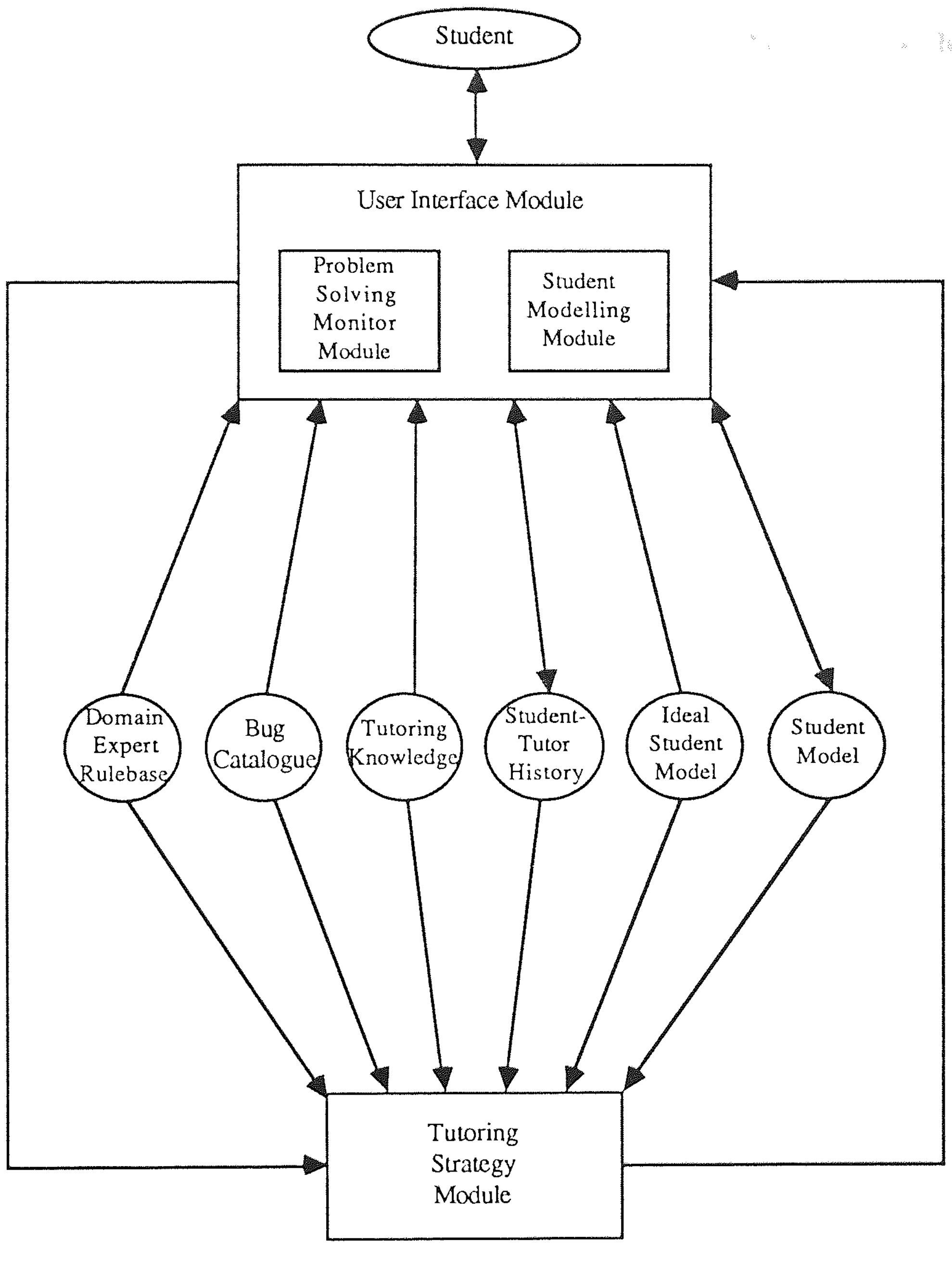
User Interface module (Problem Solving Monitor module to be exact) has presented a problem to a student and he/she is judged by FITS to be finding difficulty with it, the by-then-invoked Tutoring Strategy module may instruct the User Interface module to do one of the following:

1. Provide a solution to the stage where he/she is finding difficulty.
2. Provide a diagnosis of his/her misconception.
3. Provide a hint.
4. Provide an example.
5. Ask a simpler question in order to diagnose misconception.
6. Take the student through a Socratic (learn-along) dialogue where he/she is asked leading questions which eventually results in him/her knowing what to do.
7. Teach/reteach the concept/task by exposition of relevant course material along with appropriate examples. Procedural concepts/tasks are taught such that the student can learn the steps involved, the sequence of those steps, and when the concept/task is applicable. Factual concepts are merely presented as such.
8. Provide a Vanlehn-type teaching sequence in a situation where the student is deemed not to know where and when to use the skills/concepts that he/she has demonstrated to have acquired.
9. Move on to test a different skill/concept or present more difficult problems.

When all the concepts/tasks are deemed to have been understood to a required level of proficiency, as reflected by the student model, the Tutoring Strategy module will then instruct the User Interface module to terminate the tutoring of the particular student by FITS. However, if the student left halfway, this will be reflected in the student model and on resumption, the Tutoring Strategy module will again be invoked which will decide on what is to be done; this highlights how important student modelling is to the Tutoring Strategy module. In fact, Self (1988b) maintains that the student model and the Tutoring Strategy module should be constructed in tandem. However, the Tutoring Strategy module consults or makes use of information from many other data modules apart from the student model, e.g. student-tutor history, bug catalogue, ideal student model, etc. This is apparent in FITS's overall architecture which, along with descriptions of these data modules, is presented in the next section. Nevertheless, the Tutoring Strategy module remains static, i.e. its strategies are never enhanced as in the case of self-improving tutors, e.g. the QUADRATIC tutor (O'Shea, 1982a).

### 6.3.2 Data Modules

Figure 6.2 presents the overall architecture of FITS. This section mainly describes the major data structures implemented (see Figure 6.2). (The other components have already been described in the previous section.)



▭ Active Components ○ Data ⟶ Flow of information

Figure 6.2 - FITS's Overall Architecture

Figure 6.2 shows that there are six major data structures in FITS. The functions of these data modules are next described.

The *Domain Expert Rulebase* contains all the correct rules used for solving all fraction addition problems. These rules are sometimes referred to in the literature as 'bon-rules' as opposed to 'mal-rules'; these terminologies appear to have French foundations. These rules are used by the Problem Solving Monitor module to monitor (and sometimes solve) any fraction addition problem which the student may be attempting. This is a domain-dependent data module.

The *Bug Catalogue* is a library of the common misconceptions or bugs for the various operations involved in fraction addition. These mal-rules are used by the Problem Solving Monitor to diagnose common misconceptions in the student's understanding. It never changes, i.e. new bugs are never discovered by FITS, though it could easily be added to. Such a catalogue is important because students' bugs must have an important role in guiding the system to choose the best strategy (Aiello *et al.*, 1988); hence it is also used by the Tutoring Strategy module. The bug catalogue is also a domain-dependent data module.

The *Tutoring Knowledge* is the *what* to be communicated or taught to the student (i.e. rules that represent the subject expertise). Hence, it contains explicitly represented knowledge about the structure of fraction addition in terms of the relationships between various concepts and sub-concepts or between various tasks. This domain-dependent data module's contents include the following:

1. A syllabus (concept/sub-concept or task relationships).
2. A list of sub-concepts (including text and menu options) associated with each concept.
3. A list of tasks (including text and menu options) associated with each sub-concept (tasks are the basic units in the syllabus and do not sub-divide any further).
4. A piece of exposition for each concept/task.
5. An example exposition template for each concept/task into which generated examples for the concept/task will be 'plugged in'.
6. Some explanation for possible wrong answers to various concepts/tasks.

7. Some quiz questions for each declarative concept/task (the procedural or numerical questions are generated).
8. Correct answers to declarative questions (answers to procedural questions are also generated).
9. Knowledge of prerequisite relationships amongst concepts, sub-concepts and tasks.

The Tutoring Knowledge is implemented as part of the FITS system and does not change over sessions, i.e. it is static. It is used by the Tutoring Strategy module when deciding whether the Problem Solving Monitor module should teach/reteach some concepts/tasks as well as quiz or test the student's knowledge on the taught/retaught concepts/tasks.

The *Student-Tutor History* is an explicit and dynamic chronological record of the interaction between the student and FITS. It records the concepts/tasks presented, the number of times a concept/task has been presented, the concepts/tasks tested, some indication of whether a question on a concept/task was rightly or wrongly answered, and the mal-rules which the student has exhibited. This data module is updated by the Student Modelling module. It is initialised (emptied) at the beginning of each session (a new level of difficulty) and updated surreptitiously in the course of all interactions between the student and FITS during such a session. After the session, the contents of the Student-History data module are used by the Student Modelling module to update the particular student's student model to reflect what it now believes to be the student's new state of knowledge after the session. In other words, this data module allows FITS to 'remember' what a student has done, but this 'memory' lasts only one interactive session. For example, if the Student-Tutor History data module, after a session, reflects the fact that the student has consistently misapplied a particular rule, or he/she has misapplied the rule more often, or has consistently exhibited a mal-rule, then the

Student Modelling module will update the student model to reflect this fact. Such a scheme prevents 'one-offs' from being used to update the student model. Such information in this data module also enables the Tutoring Strategy module to:

1. Avoid re-presenting a recently given example, task or problem.
2. Determine how many responses a student has given to a particular task so as to in turn determine when to provide more help.

The *Ideal Student Model*, as its name implies, is a representation of what the desired model of the ideal student should look like. This is mainly used by the Tutoring Strategy module for overlaying, i.e. to indicate those part(s) of the syllabus (concepts or tasks) that it is believed the student lacks or to expose those concepts/tasks for which the student has not displayed an adequate level of proficiency. Such information is obviously vital for the Tutoring Strategy module.

The *Student Model:* O'Shea & Self (1983) provide an excellent broad definition for a student model as "any information which a program has which is specific to the particular student being taught. The information could range from a simple count of how many incorrect answers have been given, to some complicated data structure which purports to represent a relevant part of the student's knowledge of the subject". Therefore, the student model reflects the student's strengths and weaknesses as well as his/her coverage of the syllabus. It also contains other information such as a belief in the student's understanding of each concept/task or a belief in the student's proficiency in fraction addition as a whole. (Self (1988b) argues that student models should not pretend to reflect the student's *actual* knowledge state, for this is impossible; rather they should reflect *beliefs* concerning the student's knowledge state).

The student model is initialised when the student first logs onto FITS by statistically analysing his/her answers to a carefully designed pre-test. This provides initial beliefs on his/her state of knowledge as perceived by the Student Modelling module. It by no means pretends to be accurate; human tutors probably never have completely accurate models of their students. Subsequently, overlaying and buggy techniques are used to update and utilise the student model.

It is clear so far that the student model is a key component of an ITS as it seems indispensable if ITSs are to justify the adjective 'intelligent'. It is a vital data module for the Tutoring Strategy module to consult before deciding on what to do next. As Self (1988b) advises, the student model was constructed in tandem with the Tutoring Strategy module; hence every feature of the student model and the ideal student model is explicitly linked with its processing in the Tutoring Strategy module.

## 6.4 Control and Data Flow

Figure 6.3 shows the control and data flow within FITS's architecture. The functionality of all the various modules that constitute the architecture has already been described. This section details the operation of the architecture as a whole, i.e. it explains how FITS works. Hence, the 'inside' details of these modules as well as details of several mentioned processes, e.g. pre-modelling, modelling, etc., are omitted: in fact, they are left for the next chapter. The discussion here concerns control and data flow, and how the FITS's modules intercommunicate in operation and therefore it does not provide any protocols of FITS in use; this is also a concern of a later chapter.

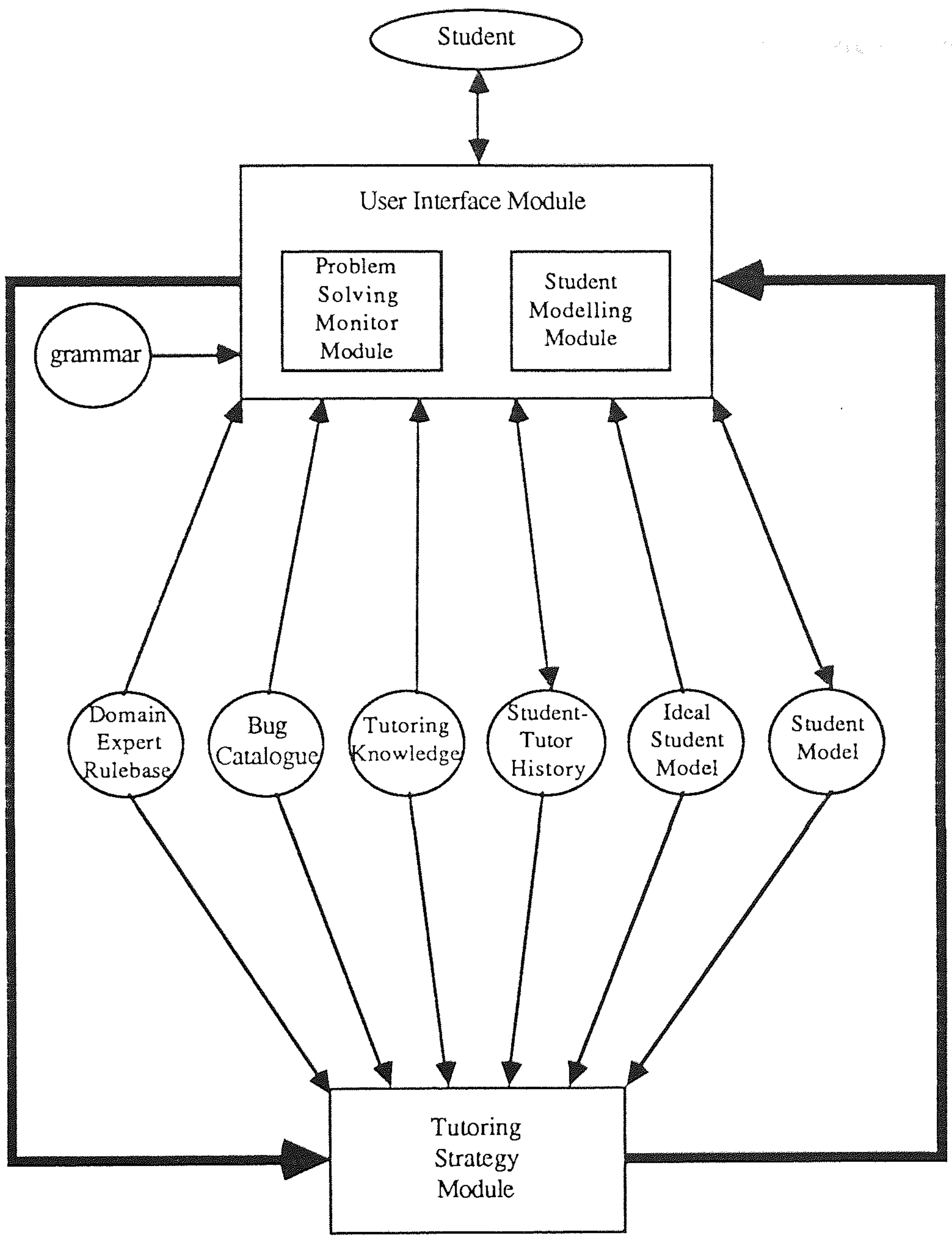


Figure 6.3 - FITS's Control and Data Flow

Conceptually, as mentioned previously, the control flow within FITS's architecture (see Figure 6.3) is between the User Interface module (i.e. Problem Solving Monitor + Student Modelling modules) and the Tutoring Strategy module. Each student-tutor cycle could involve several control flows between these two components; it also usually involves much flow of data within the architecture. These will be discussed in the ensuing paragraphs.

FITS normally commences tutoring a new student by analysing his/her results to a carefully-designed pre-test (it expects that the student's responses to these pre-test questions had previously been stored and so can be retrieved on request). The User Interface Module (the Student Modelling module to be exact) uses these responses to perform pre-modelling, i.e. it creates a unique student model for the particular student which it initialises according to the performance of the student on the pre-test. The vital role of a student model to truly individualised instruction cannot be overstated. Control is now passed to the Tutoring Strategy module which will have to decide on the next tutoring action. This module makes use of many data to arrive at such decisions, as shown in Figure 6.3. At this stage, it mainly compares (overlays) the just initialised Student Model with the Ideal Student Model; many beliefs could be inferred from this comparison, which include the following:

A. Some prerequisite skills are unknown to the student, i.e. they are deemed missing from his/her repertoire.
B. All prerequisite skills have been acquired but at least one has not yet been mastered, i.e. the student is deemed not proficient in at least one of these skills.
C. All prerequisite skills have been mastered but the student has not yet mastered the overall skill of fraction addition.
D. The student is an expert at fraction addition.

Naturally, in the case where the Tutoring Strategy module believes the student an expert (i.e. D above), it immediately returns control to the Problem Solving Monitor module instructing it to terminate tutoring the student.

If A applies, the Tutoring Strategy module normally instructs the Problem Solving Monitor module to sequentially teach the prerequisite skills that are believed missing from the student's repertoire. The latter finds an appropriate skill, presents a piece of exposition on the skill, provides adequate examples and tests the student's mastery of the skill. The Tutoring Strategy module will only be reactivated if the student has finished with the current piece of exposition. During all interaction, every input from the student is processed by the User Interface module. Its parser makes use of the fraction addition *grammar* to report immediately on any incorrect entries, e.g. syntactic errors. In the case of an incorrect solution, the Tutoring Strategy module is invoked for a decision. It may instruct the Problem Solving Monitor module to diagnose the error (using data from the Bug Catalogue). Failing this, it may instruct the latter to provide a hint and another chance for the student to try again. These decisions are made on the basis of various pieces of information, e.g. the number of attempts the student has made at this task (using data from the Student-Tutor History), the type of mal-rule that was manifested (using data from the Bug Catalogue), etc. When the User Interface module is in control, the Student Modelling module also updates the student's Student-Tutor History accordingly, e.g. noting in it the mal-rules that have been manifested, the problems he/she has attempted successfully/unsuccessfully, etc. After a *session*, the Student Modelling module uses the data in the Student-Tutor History to update the Student Model so as to reflect the student's emerging knowledge of fraction addition. A similar procedure is pursued until all the skills believed missing from the student's repertoire are taught. However, the student can quit after any successful/unsuccessful tutoring of a skill. The Student Model will reflect his/her current knowledge state, and the Tutoring Strategy module will ensure that tutoring resumes from where he/she quitted.

However, suppose that after pre-modelling, B applies (this may also apply after the tutoring of prerequisite skills that were deemed unknown to the student). Here, all the skills are believed acquired, but at least one is believed not yet mastered. The by-then-invoked Tutoring Strategy module may decide in this case that the Problem

Solving Monitor module should provide appropriate examples, drill and practice until the student is believed proficient in all the prerequisite skills. Once again, similar processes to those described in the previous chapter may take place; the student is also free to quit anytime he/she wishes.

However, in the case after pre-modelling where C applies (it may also apply after the process described in the previous paragraph), the invoked Tutoring Strategy module normally instructs the Problem Solving Monitor module to provide drill and practice on the overall fraction addition skill; at least, at this stage FITS is 'sure' that the student has mastered the required prerequisite skills. Hence, he/she is not expected to solve fraction addition problems without knowledge of them. From here normally ensues the key and interesting interactions that FITS is capable of.

The Problem Solving Monitor module generates and presents a problem of a particular difficulty level for the student to do. The student is expected to solve the problem in a step by step fashion; however, he/she is not constrained to do so. A good student (judging from his/her Student Model) is allowed to skip steps. In so doing, if he/she makes too many errors (judging from the data in the Student-Tutor History), this privilege of skipping steps is withdrawn by the Problem Solving Monitor module, i.e. the appropriate menu option is taken away. This menu allows the student communicate his/her intention prior to performing it. If the choice is legal for the stage, the Problem Solving Monitor module normally praises the student and stays quiet. Meanwhile, the Student Modelling module records an instance in the student's Student-Tutor History of correct manifestation of *where* and *when* to use the chosen operation. If on the other hand, the choice is illegal the Tutoring Strategy module is as usual invoked for a decision. Normally, the latter examines the Student-Tutor History. If it is judged good, the Problem Solving Monitor module is instructed to give another chance (it may have been a slip or a misentry). Otherwise (i.e. he/she is likely to get it wrong again, judging from his/her Student-Tutor History), the Problem Solving Monitor module will provide a hint or an example along with another chance for the student to retry. However, if the choice is again illegal, the Student Modelling module notes in the Student-Tutor History an instance of the manifestation of the student *not* knowing *when* to to apply the skill that he/she chose, as well as that which should have been chosen. In this way, those skills which the student has demonstrated to possess but which he/she does not know when and where to apply are highlighted (this was noted as a major cause of the bugs observed in the pilot studies reported in Chapter 4). The Tutoring Strategy module also instructs the Problem Solving module to inform the student on a correct operation to perform at the stage. When the student chooses this operation from the menu, he/she is now required to perform the operation so as to produce the result to the next stage. In the case where the result is correct, the Problem Solving module praises the student and redisplays the menu for the student's choice of the next operation to be performed. The Student Modelling module also notes in the Student-Tutor History an instance of that skill having been correctly performed. If on the other hand the answer to the stage is wrong, the Tutoring Strategy module may decide that the Problem Solving Monitor module should diagnose, hint, provide an example, etc. If the student's second attempt is again wrong, the Problem Solving Monitor module will be instructed to provide the answer in order to avoid bringing tutoring to a complete halt. Naturally, the Student-Tutor History is updated to reflect an instance of the particular skill not being correctly performed. When the fraction addition problem is eventually solved, FITS displays the student's solution and comments on its efficiency. It gives extra praises to the student who solves the problem in fewer number of steps than it would have done. If however the student's solution is long-winded, it also displays its more optimal solution so as to enable the student to improve on his/her efficiency.

Each *session* requires the student to attempt three problems of the same difficulty level so as to ensure adequate skill refinement. After the session, the

Student Model is updated with the contents of the Student-Tutor History as mentioned earlier. At this stage the model may reflect one or more of the following beliefs:

1. Some skills are still not acquired by the student.
2. Some skills are still not well mastered.
3. The student still does not know *when* to apply certain skills in his/her repertoire.

If, e.g. (3) applies, the Problem Solving module would provide a Vanlehn-type teaching session which aims at communicating knowledge of where and when to apply the acquired fraction skills. Examples and testing ensue as described previously. After appropriate reteaching and successful testing, the Problem Solving Monitor module is instructed to present the same session to the student again (i.e. problems of the same difficulty level that he/she failed to do, and hence warranted some reteaching).

In the case where the student progressed successfully (which may only be after some reteaching), the Tutoring Strategy module would now decide that the Problem Solving Monitor module should present problems of a one-higher difficulty level, i.e. move to a one-higher session. Once again, the student can quit after any session but on resumption, tutoring will continue from where he/she quitted. If the student goes through the *six* sessions successfully, his/her Student Model will ultimately become identical to the Ideal Student Model, at which stage FITS's tutoring of the student terminates (however, a student could still go through all the sessions without attaining 'perfection').

## 6.5 Some Comments on FITS's Architecture

It is important, after this discussion of how FITS works, to highlight the clear separation of domain-dependent and domain-independent parts achieved with the architecture of Figure 6.3. In this respect, it is envisaged that modifying the present tutor to teach another domain, say fraction subtraction, should not be much of a problem. The modules that require changing are evident. It is clear that the *grammar* for fraction addition will have to be substituted by that for fraction subtraction, while the parser will remain intact. The Student Modelling and the Tutoring Strategy modules will also require no changing. Admittedly, the Problem Solving Monitor module requires more work, but it also is largely domain-independent. Naturally, data modules such as the Domain Expert Rulebase, the Bug Catalogue and much of the Tutoring Knowledge will need substitution. The form of the Student Model, Ideal Student Model and Student-Tutor History will largely remain the same; however, it is clear that fraction subtraction concepts will have to replace the present fraction addition ones.

FITS's architecture largely subsumes Anderson's as well as O'Shea *et al.*'s (1984) five ring architecture (see Section 2.4.2). It also appears to subsume Hartley & Sleeman's (1973) design except for the means-ends guidance rules (i.e. knowledge of how to apply tutoring knowledge). Hence, theoretically, a system based on such an architecture should possess most of the advantages of the above-mentioned architectures. It is very interesting that this architecture resulted, considering that it was by no means pre-designed to be as such; rather, the specification of FITS presented in previous chapter led to such a design. It is left for a later chapter to reveal how far this architecture lives up to these theoretical advantages.

## 6.6 Summary

This chapter has overviewed FITS: it has presented the architecture of the ITS and described its key modules. It has also provided a description of how FITS works along with some comments on its architecture. The next chapter addresses the design details of its constituent modules.

# Chapter 7

# System Design

## 7.1 Introduction to System Design

In this chapter, the design details of the Fractions Intelligent Tutoring System (FITS) are described; Chapter 6 mainly overviewed it. Figure 7.1 depicts FITS's overall design as presented previously. The strategy adopted in this chapter is to single out every module of this architecture in turn, and discuss its design details. It is non-trivial, and also of doubtful value, to describe every single detail of these modules; rather, important aspects such as their major knowledge engineering techniques, e.g. their knowledge representation or specific Artificial Intelligence (AI) techniques and algorithms, are described. Besides, there is not much to some of them: in these cases, example rules are provided (all the modules are essentially rule-based). Where deemed necessary, some of the descriptions are provided in some pseudo-declarative code or, sparingly, in Prolog, the implementation language of FITS: Prolog's inherent descriptive, or *declarative*, as well as recursive view seems sometimes very appropriate.

It is expected that a knowledgeable reader should be able to largely re-create FITS after having read the previous chapter as well as this one. More subtle details ('tricks of the trade') can always be obtained by consulting FITS's source code. (A reader not interested in its details could also skip this chapter.)

## 7.2 The User Interface Module

It must be obvious now that this module controls the interaction between the student and FITS. It has also been mentioned in the last chapter that the student interacts with FITS mainly in terms of integers (e.g. a menu option (1,2,...)) or by fraction expressions (e.g. 2: $^1/_2$ + 3: $^1/_4$, $^1/_5$, 1 + 2 + $^1/_2$, etc). It is hence clear from the latter sentence that the language of communication between FITS and the student is very restricted.

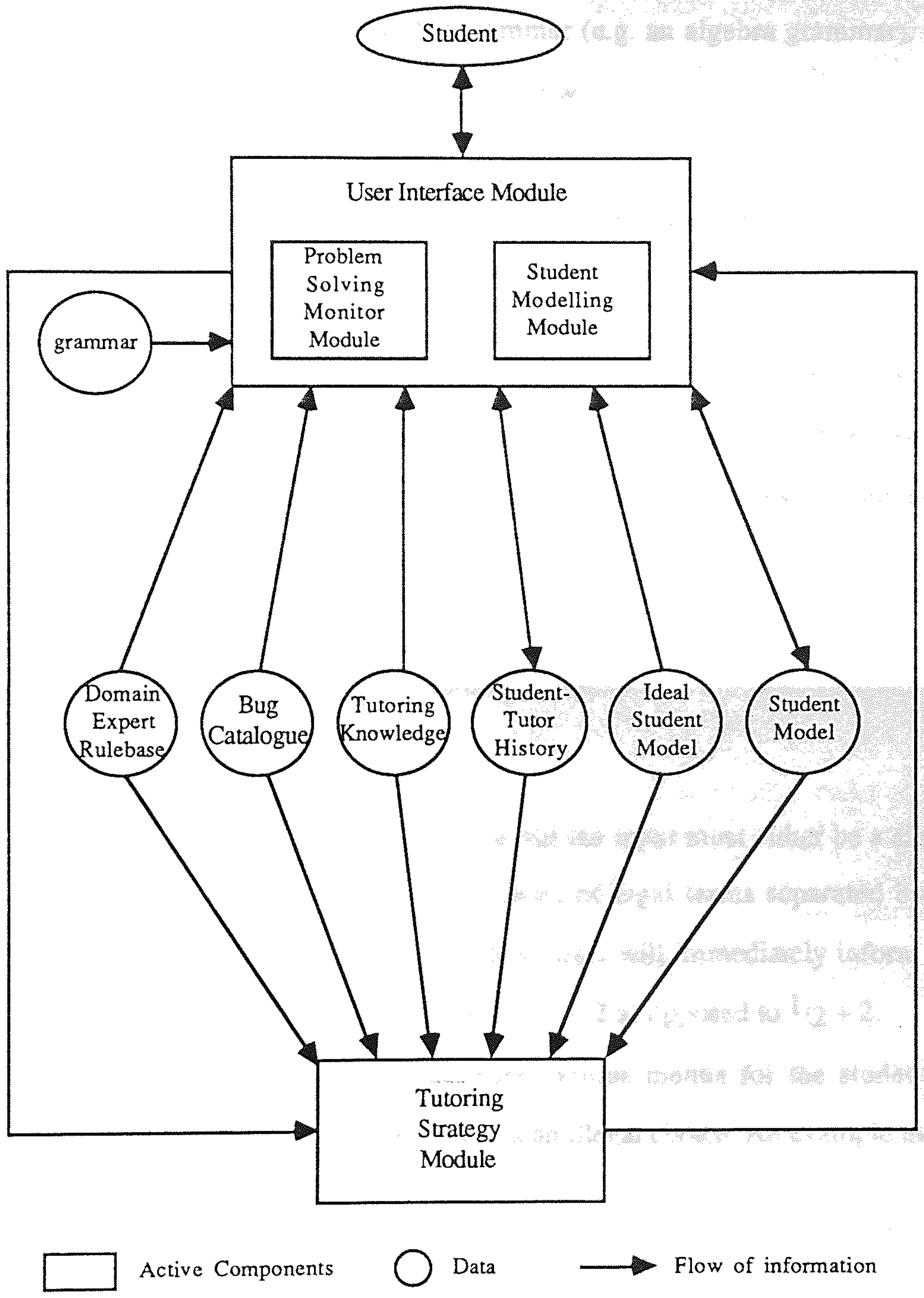


Figure 7.1 - FITS's Design

Therefore, the *grammar* (rules of the syntax of the language) is also very limited. Consequently, it is trivial to construct a parser (deterministic or non-deterministic) for such a 'language'. FITS's parser is thus a rudimentary non-deterministic one. However, the rules of the grammar are explicitly represented. They are separate from the parser (rules that actually perform the parsing) so that they could easily be replaced by another grammar (e.g. an algebra grammar). An example of a rule in FITS's grammar is given below.

```
term(N/D):- integer(N), integer(D).
term(W:N/D):- integer(W), integer(N), integer(D).
term(W):- integer(W).
term(_):- fail.
```

The above description discloses that an acceptable or legal term of the language is either of the form $^{N}/_{D}$ (e.g. $^{1}/_{2}$), W: $^{N}/_{D}$ (e.g. 1: $^{1}/_{2}$) or W (e.g. 2), where W, N and D are all positive integers. An example rule of FITS's simple non-deterministic parser is also given below.

```
parse_input(Input):- term(Input).
parse_input(First + Rest):- term(First),
    parse_input(Rest).
parse_input(_):- fail.
```

This simple recursive description defines that the input must either be a single legal term of the language (e.g. $^{1}/_{2}$) or a series of legal terms separated by '+' operators (e.g. $^{1}/_{2}$ + 1 + 1: $^{1}/_{4}$). Hence, such a parser will immediately inform the student on receiving an invalid input such as $^{1}/_{2}$ + + 2 as opposed to $^{1}/_{2}$ + 2.

The User Interface module also provides various menus for the student to choose from. It also reports on mistakes such as an illegal choice. An example menu which this module provides is the following:

```
List of Operations
1.   Add equivalent fractions
2.   Cancel fraction
3.   Find equivalent fractions
4.   Sum whole numbers
5.   Change improper fraction to a mixed number
6.   Change mixed numbers to improper fractions
7.   Rewrite fraction
8.   Provide some intermediate/final answer
WHICH<1..8>
```

As mentioned in the previous chapter, there is no autonomous User Interface module in FITS: the Problem Solving Monitor and the Student Modelling modules, amongst them, perform the role of the User Interface in addition to their own separate roles. It is therefore natural that the design details of these two modules are described next.

### 7.2.1 The Problem Solving Monitor Module

This is the monitoring module of FITS which supervises the student through the stages of a fraction addition computation. As with most problem solving, fraction addition can be viewed as centred around a general scheme, called *state (search) space* for representing problems. A state space is a graph whose nodes correspond to problem solving situations, and a given problem is reduced to finding a path in this graph. Figure 7.2 presents a section of state space for solving the fractions problem 1: $^1/_2$ + 2: $^1/_4$.

Intermediate steps (which also include mal-rule or buggy steps) and possible moves form a directed graph (state space) such as that depicted in Figure 7.2. The nodes of the graph (network) correspond to possible intermediate stages including mal-rule ones, and the arrows correspond to legal transitions between states (mal-rule transitions are legal for a single step only, after which the student is directed back onto a correct path). The Problem Solving Monitor's central role is therefore supervising the student through a path between the given initial problem (start node) to the final answer (goal node). Such supervision takes the form of commenting on the student's attempts, providing hints, examples, help, etc.

As Figure 7.2 reveals, there is an infinity of states within such a state space, hence the figure is no way near complete: there can be numerous more arrows between states in the search space shown in Figure 7.2. Therefore, it is akin to an infinite state machine (Minsky, 1972). A simplified top-level description of the main problem solving strategy is presented. The recursively executed strategy, which operates on some intermediate/final answer, is articulated in the ensuing condition-action rules, as in O'Shea (1979):

```
1         (Final_correct_answer) -> Congratulate the student and
                    stop
2         (Some_correct_intermediate_answer) -> Fetch some new
                    intermediate answer
3         (Some_incorrect_intermediate_answer) -> Diagnose,
                    remedy, hint, help, etc, and then fetch
                    some new intermediate answer
```

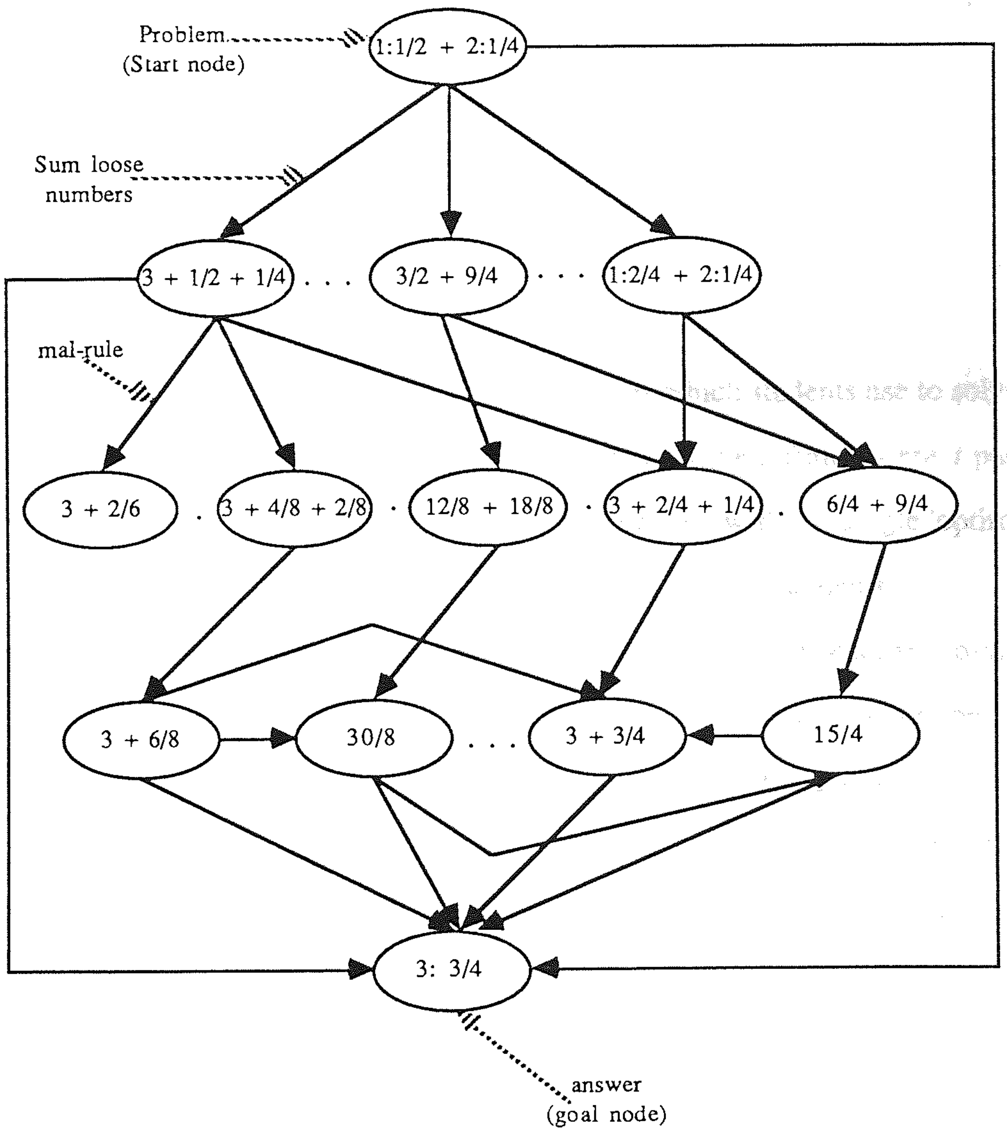


Figure 7.2 - Section of State Space for Solving the Problem 1: $^1/_2$ + 2: $^1/_4$

The above top-level recursive strategy elucidates that if the student provides the final correct answer, the Problem Solving Monitor module congratulates him/her. In the case where a correct intermediate answer is provided, some new intermediate step (which may be the final answer) is sought from the student by demanding his/her intentions and then asking him/her to perform this intention to achieve the new intermediate step. In the case of an erroneous halfway result, the Problem Solving Monitor module attempts to diagnose or comment on this step and may even provide more help (e.g. a suitable example). Afterwards, it gets the student to try again, hopefully with more success. However, there is a limit of two tries after which the student is presented with some form of diagnosis, or help, so as to avoid bringing the problem solving process to a complete halt.

Such a scheme allows for an infinite number of possible paths (solutions) of various lengths from the problem to the final answer: hence, the Problem Solving Monitor module supports various idiosyncratic means which students use to solve a fraction addition problem. (The search space represents the infinite *correct* paths (solutions) which a student might use.) This is contrasted with the single 'optimal' path on which Andersonian model-tracing tutors constrain their students.

Since mal-rule states are also incorporated in the state space, it becomes possible for the Problem Solving Monitor module to easily diagnose and report on any common systematic errors the student makes using the bug catalogue. For instance, Figure 7.2 shows a state transition from $3 + {}^{1}/_{2} + {}^{1}/_{4}$ to $3 + {}^{2}/_{6}$: the Problem Solving Monitor module immediately detects this as a manifestation of the most common fractions addition mal-rule: 'add both numerators and both denominators to obtain the numerator and denominator of the sum fraction respectively' (see Appendix D). It will also report on any illegal moves which the student may attempt to make (e.g. he/she may attempt to sum whole numbers at the stage ${}^{1}/_{2} + {}^{1}/_{3}$).

The Problem Solving Monitor module is also capable of solving all fraction addition problems (provided the number of arguments is limited to two). It does this by traversing the state space for the problem using the most optimal path; this necessitates using the most optimal transitions between intermediate steps. Hence, problem solving can be viewed as identifying an optimal list of productions (rules), and then using it to solve the problem. For example, an optimal list of productions to solve the problem 1: $^{1}/_{2}$ + 2: $^{1}/_{4}$ is as follows:

```
[sum_loose_numbers,
find_equivalent_fractions,
add_equivalent_fractions,
rewrite_fraction]
```

These rules will be used by the earlier-mentioned Problem Solving Monitor module to solve the problem. The resulting solution trace will be as follows:

```
Solving problem ->                1: 1/2 + 2: 1/4
Sum whole numbers ->              3 + 1/2 + 1/4
Find equivalent fractions ->      3 + 2/4 + 1/4
Add equivalent fraction ->        3 + 3/4
Rewrite fraction ->               3: 3/4
```

There are other capabilities which the Problem Solving Monitor module displays. For instance, it is capable of generating fraction addition problems for a required level of difficulty to present to a particular student. The following predicate

```
generate(Level_of_difficulty, Limit, Problem)
```

will use a random number generator to generate a problem/task of a required level of difficulty. The limit parameter represents the maximum allowable integer that can be generated. For example, invoking

```
generate(task_level_1, 8, Problem)
```

might generate the problem $^{2}/_{7}$ + $^{1}/_{8}$ which is instantiated to *Problem*. Student generated problems are also allowed for.

Other capabilities include its ability to compare the student's solution to that of the Problem Solving Monitor's so as to enable the student to learn more optimal ways of solving a problem, especially if his/her solutions are long-winded. It is also capable of testing the student's understanding of a particular task.

## 7.2.2 The Student Modelling Module

It was noted in Chapter 6 that the Student Modelling module, which functions in very close collaboration with the Problem Solving Monitor module, has two main functions namely:

1. It initialises the student's model by using his/her answers to a carefully designed pre-test of ten questions. This process is called pre-modelling and its rationale stems from the fact that it seems unreasonable to assume that each student starts up with the same structuring of state space (Self, 1987b).
2. It subsequently up-keeps this model by maintaining a Student-Tutor history data module.

To achieve any form of modelling, the syllabus of fraction addition was divided into twenty-four concepts, sub-concepts and tasks. These are discussed in a later section. To accomplish the first of the above-mentioned two functions, a scoring scheme was devised so as to be able to assign credit/blame to the necessary concepts/tasks. Since there are twenty four concepts/tasks, every scoring vector or array comprises twenty four values. (Each scoring vector carries implicitly-stored information about knowledge of prerequisites relationships. This is further explained later in this chapter.) To score concepts/tasks, three values were used. These three values along with their significance are described next:

| | |
|---|---|
| 1 | A concept/task is scored '1' to represent *credit* assignment to that concept/task. Hence, concepts/tasks in a 24-element credit vector which are scored '1' represent credit assignments to those concepts/tasks deemed directly responsible for the correct solution to the problem. Each of the ten problems of the pre-test have a unique credit (correct) vector. |
| -1 | A concept/task is scored '-1' to represent *blame* assignment to that concept/task. A mal-rule 24-element vector contains '-1's which represent those concepts/tasks which are directly blamed for the manifestation of the mal-rule. Hence, different mal-rules have different vectors. |
| 0 | A concept/task is scored '0' to reflect that fact that knowledge (or lack of knowledge) of that concept/task is independent of solving the specific problem. Both mal-rule and credit vectors have '0's in them. |

For example, the first question of the pre-test, $^{3}/_{4} + ^{1}/_{5}$, has the credit vector shown below.

(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

This means that only 10 of fraction addition's twenty four concepts/tasks are deemed necessary for solving the problem and hence, are credited when correctly solved. A credit vector consists only of '1's and '0's. Likewise, an example mal-rule vector is given below.

(0, -1, -1, -1, -1, -1, -1, -1, 1, 1, 1, 1, 1, 1, -1, -1, -1, -1, 1, 1, -1, -1, 1, 0)

This indicates that two concepts/tasks were independent of solving the problem while thirteen concepts/tasks can be blamed for the manifestation of the mal-rule. The '1's signify that nine concepts are deemed known, even though the eventual answer is wrong.

To pre-model the student, the Student Modelling module analyses the answers to the carefully designed pre-test. It produces an optimal set of mal-rules which explains all or most of the bugs, if any, in the pre-test solutions file, as in DEBUGGY (Brown & Burton, 1978). It proceeds to score all the correct responses with their respective credit vectors. Similarly, it scores all the mal-rules with their respective mal-rule vectors. Undetected mal-rules have a default scoring vector. For example, suppose only two of the ten questions are correctly answered, and the optimal bug set yields six mal-rules, then they will be scored accordingly. However, because only eight of the pre-test problems have been accounted for so far (two correct and six buggy), then the two mal-rules that are unaccounted for are individually scored using the single default vector.

These vectors are all added together to give a sum vector which is compared to two threshold vectors to yield a normalised vector containing scalar values ranging from 0 to 2 representing the Student Modelling module's initial beliefs of the student's understanding of the twenty four concepts. An example vector of beliefs is shown below.

(1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 2, 1, 2, 0)

The '0's represent concept/tasks deemed not 'known' to the student, the '1's signify moderately understood concepts/tasks while the '2's indicate concepts/tasks that are believed to be well understood by the student. This vector of beliefs is used to initialise the student model. Therefore, this model will initially reflect a distinct lack of the five '0' concepts/tasks; hence, tutoring will probably commence by presenting these concepts/tasks along with appropriate examples. This initial model also contains beliefs of whether the student has the knowledge of *when* and *where* to use the concepts/tasks that he/she knows *how* to perform. A predicate *mastery (Level_of_mastery)* is also recorded in the student model which reflects belief of the student's mastery of addition of fractions as a whole. *Level_of_mastery* instantiates to '2' for expert mastery, to '1' for novice mastery (i.e. not skilled), and to '0' for absolutely no mastery. The use of this mastery predicate will be evident in the next section. Ford (1987a, 1987b) proposes such a scheme.

A major advantage of such a rigorous scoring scheme is that the Student Modelling module can reasonably correctly identify the fact that some concepts are 'known' or 'half-known' by the student, even though the answers given to all questions in the pre-test are wrong. It is by no means accurate: after all no human tutor has completely accurate models of his/her students. However, it is reflective enough to provide a good start for truly individualised instruction which ITSs strive to do.

Subsequent up-keep of this model involves close collaboration between this module and the Problem Solving Monitor module. A record is kept of all sessional interaction between the student and the FITS; a view of this record will be shown later. After analysing this record at the end of a session using simple statistical techniques, the student model is updated by the Student Modelling module to reflect its new 'beliefs'. This was explained at some length in Chapter 6 as well as later on in this chapter.

## 7.3 The Tutoring Strategy Module

FITS has a largely general (i.e. subject independent) rule-based tutoring strategy which decides the next tutoring action partly on the basis of what has previously happened in the session but mostly on the current state of the student model. Hence, the 'marriage' of the Tutoring Strategy module and the student model in producing truly individualised instruction cannot be overemphasised. In FITS, the tutoring strategy being used depends on what its goal is at that particular moment (e.g. is the goal concept acquisition?). Perhaps the best way to further explain this module is to examine it in this light.

At the bottom line, the ITS strives to teach students, i.e. the students are *learning* from the ITS. There is a considerable school of thought that there are two basic forms of learning, especially for mathematical domains: knowledge acquisition and skill refinement (Carbonell *et al.*, 1983). The goal of knowledge (skill) acquisition is to get the student to learn new symbolic information coupled with the ability to apply that information in an effective manner. However, it appears that the bulk of the learning process consists of refining the acquired skills by the repeated practice and by correcting deviations from desired behaviour. The latter is termed skill refinement. FITS caters for both types of learning goals; each goal has a different tutoring strategy.

It is appropriate to return to the normalised vector used to initialise the student model which comprises '0's, '1's and '2's. Clearly, the type of tutoring necessary for the '0' concepts/tasks is tutoring for concept acquisition while tutoring for skill refinement is suited for the '1's. Hence, the overall tutoring strategy of FITS begins to unfold. To further clarify this strategy, a predicate *analyse_student_model (Name, State_of_Model)* is introduced which when invoked analyses the student's student and instantiates the variable, *State_of_model*, as follows:

| | |
|---|---|
| state_0 | *State_of_model* is instantiated to 'state_0' when the analysed student model yields one or more '0' concepts, i.e. it reveals the belief that some concepts are 'unknown'. |
| state_1 | *State _of_model* is instantiated to 'state_1' if the analysed model yields no '0' concepts but reveals at least one or more '1' concepts, i.e. concepts are believed to have been acquired but at least one has not been refined. |

| | |
|---|---|
| state_2 | *State_of_model* instantiates to 'state_2' when the model yields only '2' concepts and the believed level of mastery of the overall skill of addition of fractions (as reflected by the *mastery (Level_of_mastery)* predicate) is a '1', i.e. novice mastery. In other words, all the subskills are well known but the overall skill is not yet well refined. |
| state_3 | *State_of_model* instantiates to 'state_3' when the model yields only '2' concepts and the student is believed to have expert mastered the overall skill of addition of fractions, i.e. his/her overall level of mastery is also a '2'. |

Using the state vector *(Name, State_of_model)*, FITS's recursively executed overall tutoring strategy is expressed, as in Ford (1987a, 1987b) and O'Shea (1979), in the following condition-action rules. (The '*'s instantiate to any student's name):

```
1       (*, state_3) -> Stop
2       (*, state_2) -> Tutor for overall skill refinement
3       (*, state_1) -> Tutor for skills refinement
4       (*, state_0) -> Tutor for skills acquisition
```

It is also worth recognising how domain independent this overall tutoring strategy is. It reveals that the predicates of the Tutoring Strategy module match variables whose value are actions of the production rules of the student model, as suggested by O'Shea *et al.* (1984). In other words, it shows how features of the student model are explicitly linked with their processing in the Tutoring Strategy module as Self (1988b) advises.

### 7.3.1 Tutoring Strategy for Concepts/Skills Acquistion

It is reasonably well known in the educational fraternity that the most powerful strategy for teaching someone a specific concept is to first define the concept (Self, 1987a; Tennyson & Park, 1980). In fact, Tennyson & Park (1980) propose a four-step process for concept/skill tutoring from the literature; these have also influenced the design of FITS's tutoring strategies. The steps briefly are:

1. The taxonomical structure of the content to be communicated should be determined (i.e. the syllabus has to be defined).
2. A concept definition should be prepared for each concept/skill, and a pool of examples should be selected or generated for each.
3. The examples of the various concepts should be arranged and presented so as to reveal the concepts' inter-relationships.
4. The presentation order of the testing questions should be arranged according to the divergency and difficulty level of the examples in step 3, and also according to the updated information about the learner's knowledge state.

Steps 2 and 3 constitute the core of the tutoring sub-strategies being described in this subsection. After defining the concept, FITS provides examples to reinforce these defined concepts. Some of the concepts are largely declarative (e.g. plain facts). Hence it can be argued that this sub-tutoring strategy supports many forms of learning (Carbonell *et al.*, 1983): learning by being told (concepts/tasks definition), rote learning (facts) and inductive learning from examples. In Prolog terms, this simple sub-strategy is expressed as follows:

```
tutoring_strategy_concept_acquisition(Name, []):- !.
tutoring_strategy_concept_acquisition(Name, [First_concept|
     Rest_of_concepts]):-
     exposition(First_concept),
     examples(First_concept),
     quiz(First_concept)
     tutoring_strategy_concept_acquisition(Rest_of_concepts).
```

This recursive strategy takes a list of concepts 'unknown' to the student. If this list is empty (i.e. all concepts are believed 'known') then the strategy stops. If not, it recursively teaches each concept by exposition and examples. Naturally, the prerequisites of a concept 'unknown' to the student will be tutored before the concept itself. FITS also quizzes the students on the concept/task after it teaches it. This stage is itself recursive: if the quiz reveals a lack of understanding of a concept, the strategy is reinvoked and tutoring proceeds only after the student demonstrates understanding of the concept. If the student quits, tutoring will resume at wherever he/she quitted.

After all the concepts are deemed acquired by the student, another sub-strategy called Vanlehn tutoring strategy takes over. The goal is to impart to the student knowledge of *where* and *when* the acquired concepts are used according to the student model (Chapter 4 noted that children usually use correct rules in the wrong settings. Vanlehn's (1987) felicity conditions addresses this issue: hence the name Vanlehn's tutoring strategy). This sub-strategy's goal is therefore similar to that of step 3 of the above four-step process. In FITS, this sub-strategy involves presenting fraction problems of graded levels of difficulty (the more difficult levels involve the use of more concepts/tasks). Each example introduces a new top-level sub-concept and hence highlights where, when and how these top-level sub-concepts are used. Similarly, as in the previous sub-strategy, the student is quizzed on his understanding of this knowledge; a poor showing reinvokes the strategy. In brief, the structure of the Vanlehn tutoring strategy is largely similar to the previous one. The student model is always regularly updated to reflect new beliefs.

### 7.3.2 Tutoring Strategy for Skills Refinement

The goal of this sub-strategy is to thoroughly refine individual concepts/tasks as opposed to refining the entire skill of fraction addition, i.e. to get the student from a score of '1' to a score of '2' for every necessary concept/task. This strategy involves drilling the student on the skill by providing as many examples as the student requires. Afterwards, the student is quizzed on his/her level of mastery of the concept/task and his/her model is updated accordingly. This sub-strategy is, in fact, subsumed in the tutoring strategy for concept/skill acquisition described previously: it is normally invoked after concept definition (see Section 7.3.1).

### 7.3.3 Tutoring Strategy for Overall Skill Refinement

It seems fair to say that most of the learning in mathematics comprises learning of acquired skills by repeated practice and by correcting deviation from desired behaviour, i.e. skills refinement. The strategy of tutoring for overall skill refinement, i.e. to refine the overall skill of fraction addition, is adopted in the latter stages of FITS's tutoring; it is also the most used sub-strategy and possibly the most interesting. Its goal is to provide enough refinement to the overall skill of fraction addition, which is believed to be only novice mastered, despite the belief that all the necessary concepts/tasks are well known by the student. Therefore, the strategy attempts to 'merge' these skills together primarily via repeated practice. The strategy is built around the following state vector *(Name, Task_difficulty_level, Status_of_model)*.

*Task_difficulty_level* instantiates to the current level of difficulty of the problems being tackled: there are six such levels. Each of these levels corresponds to a session: hence six sessions. *Status_of_model* instantiates to 'okay' or 'not_okay' representing a required or the deficient student model at that level respectively. This frequently recursively executed sub-strategy is expressed in the following condition-action rules.

```
1    (*, Highest_task_difficulty_level, okay) -> Stop
2    (*, Some_intermediate_task_difficulty_level, okay) ->
              Move on to a new session, i.e. a one higher task
              difficulty level
3    (*, Some_intermediate_task_difficulty_level, not_okay) ->
              Choose some tutoring strategy (action) according
              to the needs of the inadequate student model and
              then return later to this tutoring strategy at
              the same task difficulty level
```

The condition-action rules elucidate that a student is allowed to proceed to the next session if his/her model reflects an expected model at the stage. When the final session is completed and this, hopefully, becomes identical to the ideal student model, FITS terminates its tutoring of the student (each session consists of three problems of the same difficulty level which the student solves under the 'watchful eye' of the tutor). The model is updated after such a session. In the case where the model reveals deficiencies, some appropriate tutoring action is taken. It may involve re-teaching/re-presenting some concepts all over again. The student will eventually return to this strategy at the session where he/she was having difficulty. Clearly, this sub-strategy's goal is therefore similar to that of step 4 of the four-step process of Section 7.3.1. This strategy can thus invoke any of the others, if need be.

## 7.4 The Domain Expert Rulebase

This domain-dependent data module contains the bon-rules (correct rules) used by the Problem Solving Monitor module to solve all fraction addition computations. Referring back to the search space diagram of Figure 7.2, it is clear that these bon-rules basically enable the transition from one correct state to another. Therefore, the general form of each bon-rule looks like the following state vector *(Operation, Old_state, New_state)*. For example, when the bon-rule

```
(sum_loose_numbers, 2: 1/2 + 1: 1/4, New_state)
```

is invoked, the variable *New_state* instantiates to 3 + $^1/_2$ + $^1/_4$. All the bon-rules in the Domain Expert Rulebase perform similar state transition processes.

## 7.5 The Bug Catalogue

The Bug Catalogue contains all the common mal-rules for the various operations involved in fraction addition. Hence, the nineteen mal-rules of Appendix D were not just coded in; rather, each of these mal-rules was critically analysed so as to reveal at which primitive operation the error was made. In this way, other common mal-rules for the basic fraction operations were generated. For example, the analysis of the bug "the child did not enlarge properly..." (see Appendix D), reveals the culprit top-level primitive operation as 'find equivalent fractions'. This bug thus becomes a mal-rule for the operation. Such analysis was necessary since a tenet of model-tracing is to diagnose common errors as they occur. The general form of a mal-rule looks like the following state vector *(Operation, Mal-rule_number, Old_state, New_buggy_state)*. For example, when the mal-rule

```
(add_equivalent_fractions, 1, 1/4 + 1/2, New_buggy_state)
```

is invoked, *New_buggy_state* instantiates to $^3/_6$ which exposes the mal-rule "the child adds both numerators and both denominators to obtain the numerator and the denominator of the sum respectively". Such a scheme also enables only the results of the mal-rules of the particular operation being performed to be matched against the student's result, rather than comparing it with all the results of all the mal-rules in the bug catalogue.

## 7.6 The Tutoring Knowledge

This module is by far the largest of the data modules and it mostly contains all the 'raw' knowledge to be communicated to the student. Of course, the main content is the syllabus of the addition of fractions domain, i.e. the relationships between various concepts and sub-concepts/tasks. However, there are other contents including various menus, expositions, explanations, knowledge of prerequisite relationships amongst concepts/tasks and some 'equivalence' knowledge of fractions (this is explained later). Some of these knowledge is stored as *frames*; to be exact, as rules of frames (this is explained later).

### 7.6.1 The Syllabus

It is evident from the previous paragraph that the syllabus plays a very central role in FITS's tutoring. Clearly, a syllabus for the fractions addition had to be determined. For a given domain, the exact set of subskills/concepts can be extremely difficult to define (Wenger, 1987). Burton (1982) defines a subskill as "any isolatable part of the skill that it is possible to mislearn". He argues for the set of all observable bugs to be partitioned under the equivalence relation of giving wrong answers on the same problems, considering all possible test cases of problems. Then a primitive skill is an equivalence class in this partition, i.e. it is part of the skill that corresponds to bugs manifesting in exactly the same problems. This definition is beautiful because it capture subskills as the smallest units of knowledge that can be individually mislearned in a purely empirical way, independent of any representation of the 'correct' skill. Furthermore, this definition is also likely to produce much finer granularity than if one starts only with a representation of the correct version. Therefore, the common mal-rules of Appendix D, apart from their use in the bug catalogue, were also largely used to serve the purpose of defining the concepts/sub-concepts or subskills of the domain of addition of fractions. The concepts were also defined such that they were 'labellable'.

Figure 7.3 gives a conceptual overview of the syllabus and show the way in which concepts relate to each other and how they relate to other sub-concepts and tasks. Each point in the diagram is called a 'node' and each line is called an 'arc'. The relationship between the concepts and sub-concepts/tasks are shown by the arcs that link the nodes. The top node (root node) describes the whole domain of fraction addition. Arcs emanating from this root parent node go to the major children nodes - the top-level sub-concepts. Arcs from the latter go to further sub-concepts. At the bottom of the syllabus tree are the 'leaf nodes' or tasks. These are the basic units in the syllabus and do not sub-divide any further.

Such a tree structure also appears to be a simplification of the relationships possible in any syllabus and hence it is reasonably general. Admittedly, there are situations where the syllabus is a network. The syllabus of Figure 7.3 is also trivial to represent in Prolog. The full syllabus can be obtained by consulting FITS's source code.

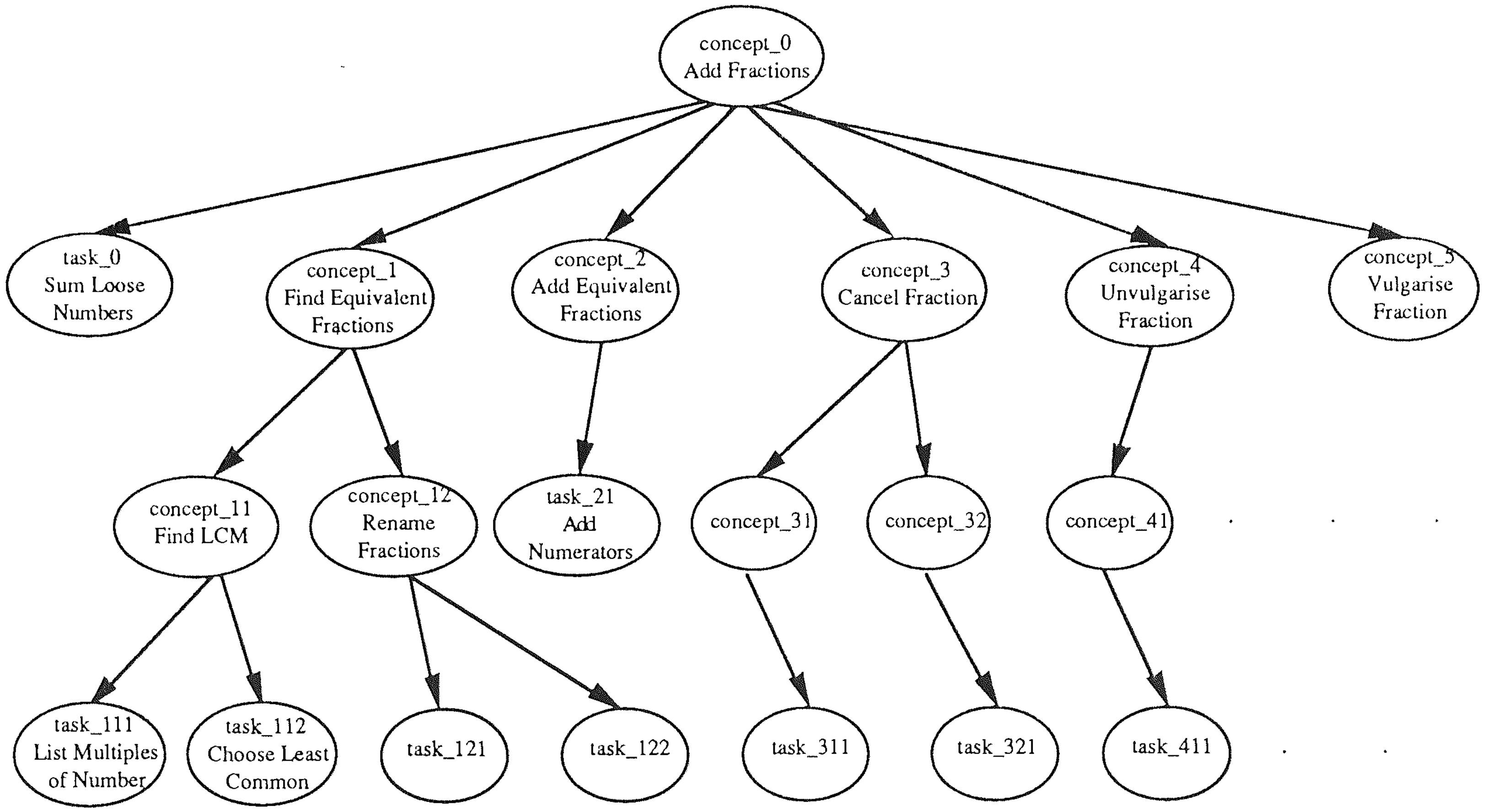


Figure 7.3 - A Section of FITS's Syllabus

### 7.6.2 Concepts/Tasks Expositions

Self (1987a) and Tennyson & Park (1980) have noted, as mentioned before, that the most powerful way of teaching someone a concept is to first define it. Each single concept/task thus has an exposition, usually between 1 and 5 lines long, which defines the concept to the student. This is what will be presented to the student if the concept is believed to be lacking after pre-modelling, and also during remediation if the concept/task is then believed, after testing, to be lacking. An attempt was made not to make the text too long or too complicated for the student to understand after first reading. An example exposition for concept_31 is provided below.

```
Find the Highest Common Factor (HCF)
The Highest Common Factor (HCF) of two numbers is the
highest or the largest of the factors common to both
numbers. Therefore, there are two steps involved in
finding the HCF of two numbers:
   1 List the factors of both numbers
   2 List the common factors and choose the highest
```

### 7.6.3 Example, Quiz and Explanation Expositions

Most of the concept/tasks definitions are further reinforced by providing as many examples as the student needs. With ready access to a random number generator, all that is further needed are templates into which generated numbers can be 'plugged in'. Naturally, the solutions to such non-declarative concepts/tasks are also generated online. Similarly, there are quiz templates for most concepts/tasks. However, the entire example and quiz expositions for the fully declarative concepts have to be pre-stored along with their solutions. Such expositions are mainly menus and their solutions mainly consist of integer menu entry numbers. There are also various explanations stored, e.g. explanations for various common mal-rules for each concept/task, explanations for minor syntactic errors, etc. There are also explanations of how concepts, sub-concepts and tasks relate to each other. A sample example which may be generated to reinforce the *concept_31* definition given in the previous section is provided below. (Questions generated to quiz the student about the concept is also similar to this example).

```
Example
Find the Highest Common Factor (HCF) of 12 and 16

Solution
The list of factors of of the numbers are:
12 -> [1, 2, 3, 4, 6]
16 -> [1, 2, 4, 8]
The list of common factors is [1, 2, 4]
Clearly, the highest common factor is 4. Therefore HCF = 4
```

### 7.6.4 Knowledge of Prerequisite Relationships

Most mathematics is considered hierarchical in nature, i.e. prerequisite skills indicate those skills a student must possess in order to master higher level skills. As in Heines & O'Shea (1985), a table of prerequisite relationship facts was drawn up, a portion of which is shown in Table 7.1.

| If the student has mastered these skills... | He/she has met the prerequisites for these skills... |
|---|---|
| NIL | task_0 task_111 task_112 and all other tasks. |
| task_111 task_112 | concept_11 |
| concept_31 concept_32 concept_41 | concept_3 |
| task_21 | concept_2 concept_22 |
| concept_41 concept_42 | concept_4 |

Table 7.1 - A Section of Prerequisite Relationships represented as Production Rules

The table should be interpreted as follows (Heines & O'Shea, 1985):

1. NIL in the left-hand column indicates that no prerequisites are required for the corresponding skills in the right-hand column. For example, no prerequisite skills are required for all the *tasks* in FITS since they are the leaf nodes in the syllabus.

2. If more than one skill is listed on a single line in the left-hand column, *all* those skills are required as prerequisites for each skill listed in the right-hand column. Thus, this is similar to the AND operation. For example, task_111 and task_112 are required as prerequisites for concept_11.
3. If more than one skill is listed on a single line in the right-hand column, *all* of the skills listed in the corresponding left-hand column are required as prerequisites for each of the skills listed in the right-hand column, e.g. both concept_2 and concept_22 require task_21 as prerequisite.
4. If a skill appears on more than one line in the left-hand column, that skill is required as a prerequisite for more than one higher level skill, e.g. concept_41 is required as a prerequisite for concept_3 and concept_4.
5. If a skill appears on more than one line in the right-hand column, any of the corresponding left-hand column provides sufficient prerequisites for that skill. This is similar to the OR function and is feasible even though there is no such example presently in FITS.

Using the full table of facts represented in a tabular form as above, the entire knowledge of prerequisite relationships can be trivially defined by production rules; hence, it is also trivial to represent in Prolog. This production rule formalism is conceptually easy to identify both the concepts for which the student has met the prerequisites and the prerequisites needed to study any particular concept. Naturally, such a representation is referenced by the Tutoring Strategy module before choosing which which concept/task to present. (It would normally present the concept's prerequisites that are 'unknown' to the student before presenting the concept itself.) This representation is also referenced by the Student Modelling module: for example, demonstrating mastery of some concept and lack of mastery of another could reveal the 'guilty' concept/task not understood by the student. For instance, suppose concepts A and B have concepts C, D, E and concepts D, E, F as prerequisites respectively. Suppose further that the student demonstrates mastery of concept A but lack of mastery of concept B. Clearly, the probable guilty concept is concept F.

### 7.6.5 Equivalence Knowledge

This equivalence knowledge refers to explicit rules which recognise the equivalence of fraction expressions. For example, the Problem Solving Monitor module should recognise that the expressions $3 + {}^1/_2 + {}^1/_4$, ${}^1/_2 + 3 + {}^1/_4$, ${}^1/_2 + {}^1/_4 + 3$, $3$: ${}^1/_2 + {}^1/_4$ and $3$: ${}^1/_4 + {}^1/_2$ are all equivalent and hence should accept any of them if it were an intermediate solution state. Similarly, if the student is at the state ${}^1/_2 + {}^1/_4$, and his/her intention is to find equivalent fractions, he/she may provide any of the following as the next resulting state: ${}^2/_4 + {}^1/_4$, ${}^4/_8 + {}^2/_8$, ${}^6/_{12} + {}^3/_{12}$, etc. These are recognised in FITS as correct competing new states. These equivalences, like the syllabus, are also trivial to represent in Prolog.

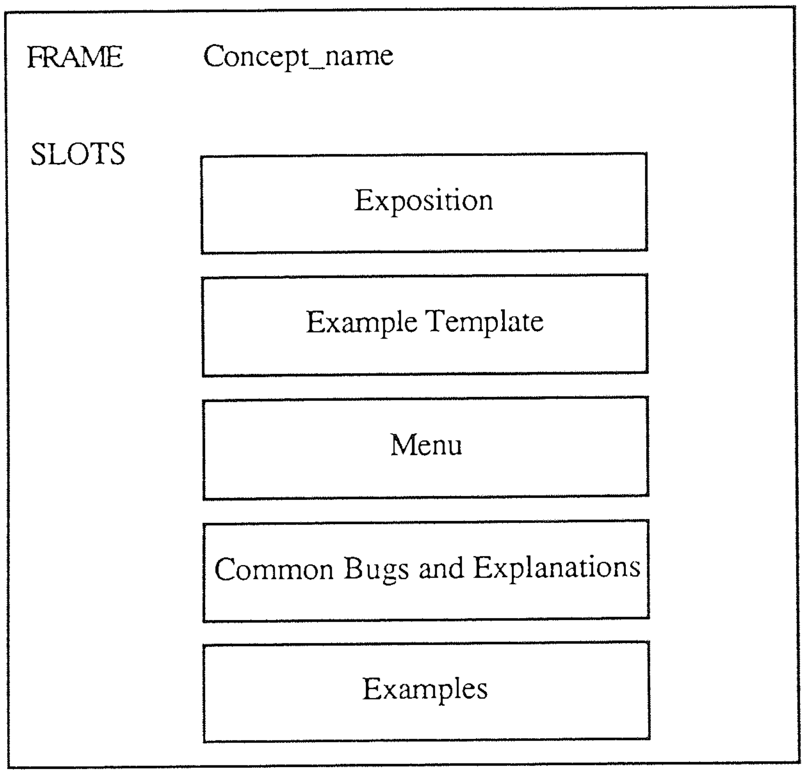


Figure 7.4 - Example Frame Skeleton used in FITS

## 7.6.6 Representation of Tutoring Knowledge using Frames

All components in FITS's architecture are largely rule-based. However, careful examination of the content of the Tutoring Knowledge clearly suggests that it could easily lend itself to a frame-based representation.

Figure 7.4 presents an example frame skeleton used for representing some of the tutoring knowledge in FITS. For every concept which uses this frame skeleton, the slots (e.g. the Examples slot or the Exposition slot) are filled with the appropriate knowledge (i.e. expositions, examples, etc.). The contents of these slots are themselves rules. In short, the Tutoring Knowledge largely comprises many rules of frames whose slots contain rules.

## 7.7 The Student-Tutor History

This data module is a dynamic chronological record of a sessional interaction between a student and FITS. It is the only data module updated during a session. A 'snapshot' of some hypothetical record of the Student-Tutor History module is provided below (the significance of the †s, ††s, §s, etc. are explained later):

```
understands_how(task_0) †
understands_when(task_0) †††
does_not_understand_when(task_0) ††
does_not_understand_when(task_0) ††
understands_when(concept_2) §
does_not_understand_how(concept_2) §§
understands_how(task_0) †
does_not_understand_when(task_0) ††
understands_how(task_0) †
understands_when(concept_2) §
does_not_understand_how(concept_2) §§
does_not_understand_when(task_0) ††
understands_how(task_0) †
understands_when(concept_2) §
does_not_understand_how(concept_2) §§
presented(1/2 + 1/4)
solved(1/2 + 1/4)
sh(concept_2, 4)
```

Figure 7.5 - Snapshot of Student-Tutor History Module

Clearly, the Student-Tutor History module consists of various sorts of records which include the following:

1. *understands_how(Concept/Task)* notes that the student has exhibited knowledge of *how* to use the Concept/Task (i.e. the procedure), e.g. *understands_how (concept_1)* indicates that the student displayed knowledge of how to find equivalent fractions.
2. *does_not_understand_how(Concept/Task)* is opposite to 1 above.
3. *understands_when(Concept/Task)* records the inference that the student has exhibited an instance of knowledge of *when* to use Concept/Task, e.g. *understands_when (concept_3)* reveals that the student has demonstrated a case of *when* to use his/her acquired cancelling skill.
4. *does_not_understand_when(Concept/Task)* is similarly opposite to 3 above.
5. *presented(Problem)* just records the fact that Problem has been presented. This prevents the re-presentation of the same problem.
6. *solved(Problem)* records that Problem has been successfully solved by the student.
7. *sh(Concept/Task, NT)* records the number of times, *NT*, that a Concept/Task has been presented to the student.

A record of a typical session of average student performance contains about a fifty such records which are statistically analysed. For example, for the hypothetical sessional record depicted by Figure 7.5, there is recurring evidence (denoted by '†'s) that the student has knowledge of *how* to perform task_0 (sum whole numbers). However, it is also evident that the student does not understand *when* to use this concept (denoted by four '††'s against the one '†††'). It is also clear that the student understands when to use concept_2 but does not understand how to perform this skill (denoted by the '§'s and the '§§'s). This 'judgemental' information is reflected by the student model when it is updated after the session.

This data module is also consulted by the Tutoring Strategy module. For example, it will look at all the problems presented or solved during the session to ensure that none is repeated except for particular pedagogical purposes. It will also help FITS 'remember' how many responses the student has given for a concept/task so as to judge when to intervene with help.

## 7.8 The Student Model

In Chapter 6, an excellent broad definition for student model was provided. Student modelling was discussed again in Section 7.2.2, as well as in Section 7.3; hence, the student model has already been much discussed. Succinctly, without being repetitive, the student model is the data module in FITS which reflects the student's strengths and weaknesses and the student's coverage of the syllabus. It contains beliefs of the student's understanding of each concept/task and of his/her proficiency in adding fractions as a whole. There are four main records in the student model:

1. *sm(Concept, Level_of_understanding)* indicates that the student is believed to have acquired the Concept/Task. *Level_of_understanding* instantiates the '0' and '1' for expert and novice mastery respectively. In the latter case, refinement is still necessary. Each concept/task has such a record, or if it does not exist, the concept/task is thus believed to be lacking and will be revealed when compared to the ideal student model. For example, *sm (task_0, 2)* reveals belief of expert mastery of task_0 while its absence indicates belief of lack of the task from the student's repertoire; *sm (concept_1, 1)* indicates belief of novice mastery of concept_1.
2. *sm_1(Concept/Task)* represents the belief that though Concept/Task is believed to have been acquired, the student also knows *where* and *when* to use this concept/task.

3. *session(Number)* records the session which the student has reached. As mentioned before, there are six sessions each corresponding to the six task difficulty levels (see Section 7.3.2). Tutoring will continue, after termination, at the appropriate task difficulty level or session.
4. *mastery(Level)* has already been mentioned previously: it records a belief of the student's overall mastery of addition of fractions.

It has already been discussed in previous sections of this chapter how these beliefs in the student model are used by the Tutoring Strategy module; in fact, it has been emphasised that these two modules should be and are closely matched to one another.

## 7.9 The Ideal Student Model

Since this data module epitomises the student model of the ideal student, its records are the same as those in the student model described formerly. As would be expected, it represents expert mastery (i.e. all '2's) of all twenty four concepts/tasks, expert knowledge of *when* to use all these concepts/tasks and finally expert mastery of the overall fractions addition skill. When compared with the student model (i.e. overlayed), it exposes such information as the concepts/tasks believed to be lacking, the concepts/tasks which the student is believed not to have mastered, etc. Once more such information is used by the Tutoring Strategy module.

## 7.10 Summary

This chapter has presented the details of FITS's design. The next chapter presents a brief 'exchange' with FITS, and goes on to appraise it in detail.

# Chapter 8

# System Evaluation

## 8.1 Introduction to System Evaluation

Chapters 6 and 7 have addressed the issue of the construction of the system with the intention that there should be enough information for others to reconstruct it or at least understand it. The intention of this chapter is to provide an appraisal (evaluation) and discussion of FITS so as to clearly reveal its research value, and its strengths as well as its shortcomings.

It is generally accepted by most ITS researchers that evaluation, of any sort, is a neglected practice. Indeed, this largely applies to the AI domain as a whole: "the develop-test-review-throwaway sequence evident in much AI research is largely counter-productive and unnecessary" (Ford, 1988). This sequence excludes evaluation, which in turn encourages unsystematic approaches to software development that of necessity curtail the possibility of the author or others refining his/her/their ideas through the developed software. This is because the program may lack structure and modularity, as well as being incomprehensible. Appraising or evaluating research makes interested readers aware of its research value in the context of their own work.

Furthermore, ultimately all ITSs that survive the prototypical phase will have to satisfy the stringent requirements of students and evaluators such as speed of response, cost, etc. However, prior to this, they will need to answer more basic questions concerning their usefulness, i.e. their ability to foster learning. Such concerns are reflected in the appraisal methods used in this chapter. FITS is appraised and discussed in the following four ways:

1. Against its specifications to determine how well it achieved its goals. In turn, this reveals in particular how successfully the second central goal of this

research has been achieved (see Section 5.2), and more generally it brings to light the overall successes and failures of the work.

2. Against a subject-independent set of questions suggested by Self (1985b) to determine how well it lives up to its prefix 'intelligent'. The questions ask about FITS's behavioural properties in a fairly general way, e.g. 'Does the system intervene if the user appears to be having difficulty?'. In providing a response to such a question, a specific account about the extent to which it does (by providing examples) and it does not (by explicitly explaining so) is provided. The answers also attempt to relate FITS's behavioural properties to the interactive-style approach on which it is based (i.e. its architecture) by stating how it does it or why it is unable to. Such a domain-independent appraisal is also an effective way of further communicating FITS's research results.
3. Against real students to reveal its limitations to potential future usability (despite the fact that FITS was designed only as an experimental tutor to test out the suggested novel interactive-style approach). This will enable researchers to identify what modifications will have to be made to FITS to make it more usable. Admittedly, most ITS researchers are rightfully cautious of prematurely introducing ITSs into real classrooms: Self (1988a) notes that "we must be careful to distinguish 'the classroom' as a generic term for a place of learning and as a term for the physical arrangement in present schools and colleges. ICAI researchers have no commitment to the latter...". This is because ITS research is still at its very early stages; researchers are, regrettably perhaps, currently more interested in the computational expression of the principles of learning and teaching than in the provision of systems for real use in present educational settings (Self, 1988a). Nevertheless, it seems inconceivable that ITS research can be done totally independent of considerations of potential use.

4. Against O'Shea *et al.'s* (1984) thirteen "pillars" of ITS design. These "pillars" represent the authors' desired features of any computer-based tutor and hence represent very general design principles for ITS construction. Such evaluation would further expose aspects of FITS's design which could be improved to make it more useful as a computer tutor.

## 8.2 Example FITS Protocol Extract

It seems unreasonable to appraise FITS in the ways suggested above without any prior introduction to an example interaction with the tutor. A brief annotated and slightly modified example excerpt from an interaction between the tutor and an actual student, Rose, is illustrated by the protocol of Figure 8.1. Comments are in italics. The protocol is also slightly stylised, e.g. the fraction $^1/_4$ really appears in FITS as 1/4.

*[The system would begin with pre-testing and appropriate pre-teaching].*

Tutor: Solve $1:^1/_4 + 2:^1/_2$.
What operation do you want to perform?
List of Operations
1. Add equivalent fractions
2. Cancel fraction
3. Find equivalent fractions
4. Sum whole numbers
5. Change improper fraction to a mixed number
6. Change mixed numbers to improper fraction
7. Rewrite fraction
8. Provide some intermediate/final answer

WHICH <1..8>? *[Demanding the student's intention/plan]*

Student: 2.

Tutor: Oh no! There is no fraction to cancel.
Try again. *[Hint].*
What operation do you want to perform?
...

Student: 4.
*[Could equally have chosen to find equivalent fractions or to change mixed numbers to improper fractions. Therefore the system can support various idiosyncratic ways of solving the same problem, hence no one 'correct' solution].*

Tutor: Fine Rose. Perform operation.

Student: $3 + ^1/_4 + ^1/_2$.

*[Could have typed in 3:$^1/_4$ + $^1/_2$ or $^1/_4$ + 3:$^1/_2$ or $^1/_4$ + 3 + $^1/_2$. Note these various possible idiosyncratic inputs are possible due to explicit fractions knowledge representation.]*

Tutor: Correct. What operation do you want to perform?

...

Student: 3.

Tutor: Fine. Perform operation.

Student: 3 + $^2/_8$ + $^4/_8$.

*[Note these idiosyncratic inputs: could also have typed in 3:$^4/_8$ + $^4/_8$ or 3 + $^2/_4$ + $^2/_4$ or 3:$^8/_{16}$ + $^8/_{16}$.]*

Tutor: Correct. What operation do you want to perform?

...

Student: 1.

Tutor: Fine Rose. Perform operation.

Student: 3 + $^6/_{16}$.

Tutor: Not Correct. You seem to have added both numerators and denominators to provide the numerator and the denominator of the fraction respectively.

*[Classic fraction error]*.

Remember you add equivalent fractions by just adding the numerators and placing this sum over the denominator. Study this example to help you. 2 + $^1/_4$ + $^2/_4$ -> 2 + $^3/_4$. *[Hint]*.

Remember your current step is 3 + $^2/_8$ + $^4/_8$.

The operation to perform is adding equivalent fractions.

Student: 3 + $^8/_{64}$.

Tutor: Still not correct. You seem to have multiplied the fractions together instead of adding.

WELL I THINK YOU CAN DO WITH SOME HELP.

Add the numerators and place the sum over the denominator for the problem $^2/_8$ + $^4/_8$.

Student: $^5/_8$.

Tutor: Not Correct. Add the numerators: 2 + 4 = ?

Student: 6.

Tutor: Correct. Place this value 6 over the common denominator 8.

Student: $^6/_8$.

Tutor: Correct. Now add this fraction sum $^6/_8$ to the whole number 3.

Student: 3 + $^6/_8$.

Tutor. Correct.

WE RETURN TO WHERE YOU WERE.

Remember you were at the stage: 3 + $^2/_8$ + $^4/_8$.

You chose to perform adding equivalent fractions.

Now with the help I have given you, Try again.

Student: 3 + $^6/_8$.

*[Note the use of the explicit knowledge represented in the system. Also note the different tutoring strategy which FITS adopts here. The system could therefore*

```
        switch to tutoring a different aspect of mathematics,
        if necessary, before returning to tutoring fractions.]
Tutor:  Correct. What operation do you want to perform?
        ...
Student:     4.
Tutor:  There are no whole numbers to sum Rose. Try again.
Student:     1.
Tutor:  There are also no equivalent fractions to add.
        Try cancelling the fraction Rose.
        [Strategy here is to tell the student what to do].
        Remember your current step is 3 + 6/8.
        [The system notes in the student model that Rose has
        shown evidence of a lack of understanding of when to
        cancel a fraction].
Student:     2.
Tutor:  Fine. Perform operation.
Student:     3 + 3/4.
Tutor:  Correct. What operation do you want to perform?
        ...
Student:     7.
Tutor:  Fine. Perform operation.
Student:     3:3/4.
Tutor:  Congratulations Rose. You have arrived at the answer.
        [After this the system would display the student's
        full answer, with a comment on its efficiency,
        followed by its 'optimal' answer.]
```

Figure 8.1 - Extract from protocol of FITS in use

This protocol will be referred to as necessary in subsequent sections of this chapter.

## 8.3 Evaluation against Specification

In Chapter 5, the features which comprise FITS's specification were listed (i.e. the principles of the iterative-style approach - the improved model-tracing approach (see Section 5.3)). In this section, FITS is appraised against this specification to determine how well it achieves its goals; this in turn reveals how successfully the second main aim of the approach has been met (see Section 5.2). Each feature of the specification is alternately examined by providing an answer to a specification feature question. Where applicable, the reactions of students who have used FITS are also mentioned. Comparisons are also made with current mal-rule and model-tracing ITSs so as to reveal the successes/failures of the iterative-style approach. This in turn reveals how successfully the overall research objectives have been achieved.

1. *Does the system pre-model the student?*

FITS does pre-model its students. In fact, it does this when the student logs on to the system for the first time. At log in, the following exchange may occur:

```
Tutor: Good afternoon. Welcome to FITS.
       Hello. What is your name?
Student:      rose.
Tutor: Fine Rose. I want to find out how much you know about
       addition of fractions. I need to know this so that I
       can decide what and how to teach you. I can do this by
       having a look at your answers to my test of ten
       questions...
       Please enter the name of your file of solutions.
Student:      rose_file.
       [Pre-modelling occurs here].
...
```

FITS appears to provide a reflective enough pre-model to provide an adequate start for individualised tuition which ITSs strive to achieve, judging from its evaluation with real students. The weaker students were pre-modelled as such and instruction always began with preteaching some skills. On the other hand, the better ones were not pretaught any skills. This contrasts with mal-rule ITSs (e.g. FRACTIONLAND, BUGGY) and with current model-tracing tutors (e.g. the GEOMETRY tutor) which do no pre-modelling at all (see Sections 3.2.2, 3.3.2).

2. *Does the system preteach prerequisite skills?*

After pre-modelling, FITS tutors all skills or concepts/tasks that are believed to be lacking from the student's repertoire. Such tutoring may also occur after a session where the student consistently exhibits lack of certain skills. Such tutoring is mainly by exposition, examples and testing. For example, the task of adding whole numbers is taught/retaught after a session (session 2) where Rose has exhibited difficulty with it as follows:

```
Tutor: You appear to be having some difficulty with this
       session. I have noticed a skill you still have not
       mastered yet. Let us revise it again.
       Please press the <RETURN> key to continue.
Student:     <RETURN>.
Tutor: SUM WHOLE NUMBERS
       Here you just sum and record the loose or whole
       numbers in the given problem.
       EXAMPLE
       Sum the loose numbers in the problem 1:1/4 + 2:1/4.
       SOLUTION
       The sum of the whole numbers is 1 + 2 = 3.
       Another example?
Student:     no.
       [Testing of the concept occurs here].
Tutor: Let us proceed to redo session 2.
...
```

Again, current ITSs of both predominant approaches do no such preteaching on the basis of pre-modelling since they do not perform the latter. They therefore proceed to start tutoring different students of different aptitudes similarly. FITS improves on this limitation. In fact, it was observed that a couple of the students who used FITS benefited from this capability: they even remarked that they liked the way it taught them some of the skills that they did not know or had forgotten (see Appendix H).

3. *Does the system provide examples (sequence of examples/lessons) such that Vanlehn's felicity conditions are satisfied?*

FITS provides a sequence of examples (lessons) which attempts to satisfy Vanlehn's felicity conditions (see Section 4.7.4). The following excerpt demonstrates how.

```
Tutor: I have arranged the examples such that you can
       understand how and when the key skills you have learnt
       are used. Each new example introduces how and when ONE
       of these skills is used. Also note how harder problems
       use more of the skills you know.
       Please press the <RETURN> key to continue.
```

```
Student:      <RETURN>.
Tutor:  EXAMPLE
        Study the step by step solution to the problem 1 + 1/2.
        SOLUTION
        Note how and when you use the skill: REWRITE FRACTION.
        Solving problem ->       1 + 1/2
        Rewriting  fraction-> 1:1/2
        Please press the <RETURN> key to continue.
Student:      <RETURN>.
Tutor:  EXAMPLE
        Study the step by step solution to the problem 1 + 1:1/2.
        SOLUTION
        Note how and when you use the skill: ADD WHOLE
        NUMBERS. You have already seen how and when to use the
        other skill.
        Solving problem ->              1 + 1:1/2
        Adding  whole  numbers  ->      2 + 1/2
        Rewriting fraction ->           2:1/2
        Please press the <RETURN> key to continue.
Tutor:  EXAMPLE
        Study the step by step solution to the problem 1:1/4 + 2:1/4.
        SOLUTION
        Note how and when you use the skill: ADD EQUIVALENT
        FRACTIONS. You have already seen how and when to use
        the other skills.
        Solving problem ->                      1:1/4 + 2:1/4
        Adding whole numbers ->                 3 + 1/4 + 1/4
        Adding  equivalent  fractions  ->       3 + 3/4
        Rewriting fraction ->                   3:3/4
        Please press the <RETURN> key to continue.
...
```

This exposition sequence could be replayed if needed and appropriate testing would ensue; obviously the pseudo-randomly generated examples will be different.

With this feature, FITS contrasts with mal-rule and model-tracing ITSs which both acknowledge the fact that students make errors in *how* they perform computations but do not also seem to accede the fact that children also make arithmetical errors because of lack of knowledge of *when* to use their acquired skills/concepts. FITS improves on this shortcoming. All the students who have used FITS remarked that they liked this Vanlehn tutoring session, which seemed to clearly demonstrate to them when and where to use the various fraction skills (see Appendix H).

4. *Does the system monitor the student step by step?*

FITS definitely achieves this. The extract excerpt protocol of FITS in use depicted by Figure 8.1 well demonstrates this feature which is clearly not achieved by mal-rule ITSs. Naturally, current model-tracing ITSs (e.g. the GEOMETRY tutor) support this, but a limitation has always been the fact that they could frustrate the good students who do not wish to go through these step by step solutions. This shortcoming is improved on in FITS as will be demonstrated later. All the students really did enjoy this monitoring capability.

5. *Does the system diagnose in a problem solving context?*

The reader is again referred to the exchange of Figure 8.1. It clearly reveals how FITS diagnoses in a problem solving context. For example, note from the protocol of Figure 8.1 how FITS diagnoses Rose's error as having multiplied the fractions as opposed to adding them. Once more, all the students liked the diagnoses: in fact, a couple were sometimes amazed at the accuracy of FITS's diagnoses. Admittedly, mal-rule ITSs like FRACTIONLAND do achieve this but not in such a problem solving and meaningful context as model-tracing tutors (e.g. FITS) do.

6. *Does the system enable the student to communicate his/her intention (plan) prior to executing them?*

FITS certainly supports this. This is a tenet of the model-tracing approach and architecture. Figure 8.1 reveals that FITS regularly comes up with the question 'What operation do you want to perform?', followed by a menu of operations. The answer to this question exposes the many situations where students attempt to perform a wrong operation at a particular stage (i.e. it reveals those concepts which the student knows *how* to perform but does not understand *when* to perform them). It also makes diagnosis/remediation easier. However, as briefly mentioned previously, FITS is also flexible enough to allow the good students to directly provide any intermediate step or the final answer as demonstrated in the following exchange.

```
Tutor: Solve 2:3/4 + 2:1/2.
       What operation do you want to perform?
       List of Operations
             1. Add equivalent fractions
             2. Cancel fraction
             3. Find equivalent fractions
             4. Sum whole numbers
             5. Change improper fraction to a mixed numbers
             6. Change mixed number to improper fraction
             7. Rewrite fraction
             8. Provide some intermediate/final answer
       WHICH <1..8>?
Student:     8.
Tutor: Go on then.
Student:     21/4.
Tutor: Correct. What operation do you want to perform?
       ...
Student:     8.
Tutor: Go on then.
Student:     5:1/4.
Tutor: Congratulations ...
```

The students enjoyed this flexibility and remarked that FITS would have been very boring without it. Naturally, mal-rule ITSs do not support such a feature. However, model-tracing ITSs do, but do not allow for the sort of flexibility demonstrated above.

7. *Does the system remediate in a problem solving context?*

The reader is once more referred to the exchange of Figure 8.1 which clearly reveals FITS's remedial capabilities. It takes many forms, e.g. diagnosing the error, instructing the student on what to do, etc. It is worth noting the socratic remedial strategy starting and ending at the statements 'Well I think you can do with some help' and 'We return to where you were' respectively. This fragment clearly further demonstrates FITS's remedial capabilities. Such a remedial strategy is not only naturally non-existent in FRACTIONLAND and other mal-rule ITSs, but also in model-tracing ITSs like the GEOMETRY tutor which mainly, successfully or unsuccessfully, attempt a diagnosis on any of the student's steps which do not match that of their ideal student models. The students also enjoyed FITS's remedial capabilities as it always came to their rescue when they were stuck, confused, etc. The socratic remedial strategy was a favourite with the weaker students.

8. *Does the system model the user?*

FITS would hardly deserve the prefix 'intelligent' if it did not do student modelling. In the example protocol provided to supplement the answer to question 2 above, the model revealed a belief in the lack of knowledge of the task, sum whole numbers, from the student's repertoire (model) after the session. Tutoring of the task ensued; this is just one of many demonstrations that FITS does model. This contrasts with mal-rule and model-tracing ITSs which perform no modelling at all (see Chapters 3 and 4) as both approaches support neither student models nor student history databases. Hence, FITS would naturally provide more individualised instruction than all the mal-rule and model-tracing ITSs mentioned in this thesis.

9. *Does the system test the student's understanding?*

FITS's tutoring could be globally viewed as testing for all sorts of goals, e.g. testing for knowledge of *how* to use a concept, testing for knowledge of *when* to use a concept, testing to determine whether the student can handle problems of a particular difficulty level (i.e. session), etc. Hence, the exchange of Figure 8.1 is a test of whether Rose can handle problems of that particular level of difficulty. A further example of testing ensues after the tutoring of the task, sum whole numbers, mentioned above (see question 1).

```
Tutor:  Let us see if you can do this one.
        Sum the whole numbers in the problem 3:3/4 + 4:1/6.
        Please have a guess if you are not sure.
Student:     7.
```

```
Tutor   Eh! That is good. Try this.
        Sum the whole numbers in the problem 6:4/7 + 3:5/9.
Student:      9.
Tutor:  Brilliant. I think you now understand how to perform
        this skill.
...
```

The system may also test at a later stage (after a Vanlehn tutoring session) whether the student understands when and where to use the above skill as follows:

```
Tutor:  Fine. Now show me that you understand WHEN to apply
        all these skills you have acquired.
        Assume you are at the stage 2 + 3/4 + 3.
        What operation would you perform next?
        List of Operations
              1. Add equivalent fractions
              2. Cancel fraction
              3. Find equivalent fractions
              4. Sum whole numbers
              5. Change improper fraction to a mixed numbers
              6. Change mixed number to improper fraction
              7. Rewrite fraction
        WHICH <1..7>?
Student:      4.
Tutor: Good...
```

FRACTIONLAND and other 'buggy' as well as current model-tracing tutors do carry out testing, but mainly on the overall skill: clearly, they can not test for a subskill possibly lacking from the student's repertoire or about when to use a particular skill (as demonstrated in the previous two protocols) since they do not have an explicitly represented syllabus as in FITS, and also since they also do not possess some of its features such as modelling, preteaching, etc.

10. *Does the system support the various idiosyncratic ways which the student might choose to solve the problem?*

FITS certainly does achieve this; this is also a major tenet of the iterative-style approach. For example, any of the following solutions is acceptable to FITS for the problem 2: $^1/_2$ + 3: $^1/_4$.

```
Solving problem ->              2:1/2 + 3:1/4
Adding whole numbers ->         5 + 1/2 + 1/4
```

| | |
|---|---|
| Finding equivalent fractions-> | $5 + {}^{2}/_{4} + {}^{1}/_{4}$ |
| Adding equivalent fractions -> | $5 + {}^{3}/_{4}$ |
| Rewriting fraction -> | $5{:}{}^{3}/_{4}$ |

OR

| | |
|---|---|
| Solving problem -> | $2{:}{}^{1}/_{2} + 3{:}{}^{1}/_{4}$ |
| Finding equivalent fractions-> | $2{:}{}^{4}/_{8} + 3{:}{}^{2}/_{8}$ |
| Adding equivalent fractions -> | $2 + 3 + {}^{6}/_{8}$ |
| Adding whole numbers -> | $5 + {}^{6}/_{8}$ |
| Rewriting fraction -> | $5{:}{}^{6}/_{8}$ |
| Cancel fraction -> | $5{:}{}^{3}/_{4}$ |

OR

| | |
|---|---|
| Solving problem -> | $2{:}{}^{1}/_{2} + 3{:}{}^{1}/_{4}$ |
| Change mixed numbers to improper fractions-> | ${}^{5}/_{2} + {}^{13}/_{4}$ |
| Finding equivalent fractions -> | ${}^{10}/_{4} + {}^{13}/_{4}$ |
| Adding equivalent fractions -> | ${}^{23}/_{4}$ |
| Change improper fraction to a mixed number-> | $5{:}{}^{3}/_{4}$ |

These are certainly not the only solutions to this problem that FITS would accept. Clearly, there are a multitude of other correct solutions; in fact, the students who have used FITS were observed using some of the above as well as other solutions and were pleased with this flexibility. This contrasts with the single 'optimal' solutions which Andersonian model-tracing tutors support. Mal-rule ITSs do not support such a feature. FITS also has the capability of comparing the student's solutions to that of the Problem Solving Monitor module's, with a comment on its efficiency. It can also provide a solution to any problem the student may present to it. For example, the following protocol demonstrates that FITS solves 'hard' problems which are not easily solved by human experts (This was a favourite activity amongst the students who have used FITS).

| | |
|---|---|
| Solving problem -> | $67{:}{}^{567}/_{765} + 75{:}{}^{987}/_{789}$ |
| Adding whole numbers -> | $142 + {}^{567}/_{765} + {}^{987}/_{789}$ |
| Finding equivalent fractions-> | $142 + {}^{149121}/_{201195} + {}^{251685}/_{201195}$ |
| Adding equivalent fractions -> | $142 + {}^{400886}/_{201195}$ |
| Cancelling fraction -> | $142 + {}^{44534}/_{22355}$ |
| Changing to mixed number -> | $142 + 1 + {}^{22179}/_{22355}$ |
| Adding whole numbers -> | $143 + {}^{22179}/_{22355}$ |
| Rewriting fraction -> | $143{:}{}^{22179}/_{22355}$ |

11. *Does the system explicitly represent knowledge?*

As previous chapters have highlighted, FITS explicitly represents its knowledge, e.g. its tutoring strategies' knowledge is largely independent of the domain-dependent knowledge to be communicated to the student (see Section 7.6). Hence, the domain-independent modules could largely be reused for some other domain; this is a similar concept to that of 'shells' in expert systems parlance. FRACTIONLAND and most mal-rule ITSs do not have any domain-independent modules (i.e. the possible domain-independent knowledge is 'hard-wired' within the rest of the code in such systems). Admittedly, some of such tutors have begun incorporating such explicit representational schemes e.g. PIXIE (Sleeman, 1987a). Current model-tracing tutors also suffer from this shortcoming: in fact, as mentioned in earlier chapters, Anderson & Skwarecki (1986) have acknowledged this fact, and are currently devising a more general architecture for their tutors. FITS definitely seems to improve on mal-rule and model-tracing ITSs as regards explicit representation of knowledge.

12. *Does the system motivate and support a more flexible style of tutoring?*

FITS's style of tutoring is much more flexible because of all the various enhancements on the model-tracing paradigm, e.g. it supports various idiosyncratic solutions, it accepts various idiosyncratic inputs, it supports various tutorial strategies, etc. For the good students, FITS was observed to play more of a mentor role while for the weaker students it played a more collaborative/helpful role: this is akin to Barzilay's (1985) SPIRIT system. To this degree, it is flexible. However, it possibly falls short on motivation. The hope in FITS is to get the student to interact more in order to reduce the possibility of boredom. Such factors as boredom and motivation are probably best addressed by providing a much more intelligent interface. However, as demonstrated by the protocols observed so far, FITS clearly improves on mal-rule ITSs which tend to be very monotonous and uninteresting as their style of tutoring never changes (see Section 4.4.2). It also appears to improve on model-tracing tutors as they seem to be very authoritative and dogmatic in style (Yazdani, 1986a), and their tutoring is also inflexible. Nevertheless, some model-tracing tutors (e.g. the GEOMETRY tutor) provide much more intelligent interfaces as this is a main tenet in Anderson's monitoring approach. In summary, FITS does appear to have improved on mal-rule and model-tracing ITSs as regards flexibility in its style of tutoring but falls well short of some of the latter as regards motivation mainly due to its fairly 'primitive' interface by today's standards. However, as will be later discussed, the students who have used it found FITS motivating enough.

13. *Does the system maintain control over the whole tutoring endeavour?*

FITS certainly does maintain control; this is demonstrated by all the exchanges observed so far (e.g. Figure 8.1). Too much control leads to authoritarian ITSs. However, FITS is less authoritarian in approach than current model-tracing ITSs mainly because it does not employ a tutoring strategy which constrains the student to follow some 'ideal' model.

14. *Does the system provide an environment in which the interaction between it and the student is close to the reality of the day?*

FITS's monitoring capability allows for the students solving problems largely as they will with pencil and paper than current tutors do. For example, they type in the intermediate steps until they arrive at the answer, they make careless syntactic as well as logical errors just as they would with pencil and paper. This contrasts with only the final answers presented to mal-rule ITSs and the much help provided by model-tracing tutors like ALGEBRALAND (see Section 3.3.1). In other words, the 'training wheels' problem (see Section 3.3.2) would be less for a system like FITS.

This highlights the fact that some of the features of the iterative-style approach conflict: for example, providing more motivation possibly best requires more intelligent and graphical interfaces (e.g. WIMP interfaces), but this in turn increases the 'training wheels' problem. FITS, unarguably, presents a compromise. However, the interface could still be much improved with little increase in the 'training wheels' problem. In any case, a couple of the students who have used the system were, afterwards, observed solving fraction addition problems in the same manner as they were with FITS; hence, it appears FITS has largely achieved this feature.

15. *Does the system provide a transparent view of the approach on which it is based?*

This question probes whether FITS exhibits or displays all the above-questioned features. While it is hard to be objective about such a question, it appears that FITS clearly exposes most of these features. On this account, FITS seems easy to understand since the principles on which it is based could be largely identified or inferred by observing it in operation. Model-tracing and mal-rule ITSs are also transparent but are not based on as many principles as FITS.

It is thus evident from the above responses to the various questions that FITS has largely met its requirements specification as presented in Chapter 5. In the next three sections, FITS is appraised in different ways; doubtless, they will reveal some limitations in FITS's design.

## 8.4 Evaluation against Real Students

FITS, despite the fact that it was designed only as an experimental tutor to test out the proposed iterative-style approach, has also been evaluated or tested against real students to reveal its strengths and limitations for future potential usability. Four students aged between 10-12 years have used FITS though it was designed for students in the age range 12-15 (the students who were involved in the pilot studies were of this age range). All the students had some prior knowledge of fraction addition just as in the pilot studies, albeit limited. They also had to come into the department since the costly SUN machine on which FITS was developed could not be taken out. It should also be clarified that a larger evaluation (i.e. with more students) would have been preferable but there were practical difficulties.

Initially, the students were given a pencil-and-paper pre-test of ten specially-designed fraction addition questions which they completed, on average, in about 12 minutes. Their answers were fed into the computer as these results were to be used by FITS for pre-modelling. They were next presented to FITS on a SUN-360 terminal at which they spent an average of an hour; they were given some help if confused or stuck, mainly at the beginning. The students were encouraged to voice their opinions as they worked along with the tutor. After using FITS, they were presented with a blank sheet of paper to note some more of their comments in writing (see Appendix H). Lastly, they were presented with a post-test which was the same as the pre-test they had earlier attempted. The table below shows the students' scores (all scores are out of ten).

| Student | Pre-test Score | Post-test Score |
|---|---|---|
| Student 1 | 8 | 10 |
| Student 2 | 8 | 10 |
| Student 3 | 4 | 8 |
| Student 4 | 5 | 9 |

Table 8.1 - Pre-test and Post-test Results

Clearly, Table 8.1 reveals that the children learnt from their encounter with FITS: some seemed to acquire some knowledge from FITS while the others mainly improved on their fraction skills (this is clearly observed by comparing the pre- and post-test scores). Perhaps more significantly, the children, to the delight of the author, were observed solving the post-test questions in a similar systematic fashion as they had been doing earlier with FITS, i.e. going through similar sorts of steps, especially for the more difficult questions. For instance, Student 4 who seemed to have a very unsystematic and error-prone style when solving the pre-test questions was observed using FITS's more systematic style to do the post-test ones. It thus appears that FITS achieved its goal of minimising the 'training wheels' problem (i.e. FITS's style is reasonably close to that of pen-and-paper): this feature was designed into the interactive-style approach.

Generally, all the students enjoyed using the program and were very enthusiastic about aspects of it. The author's initial fears that FITS would be extremely difficult for students to use (mainly due to its interface) were proven to be totally wrong: apart from the few occasions when they needed help (mostly initially), they were observed 'plodding away' with FITS while the author carefully made notes on their commentaries. They largely expressed the opinion that they had learnt something from and/or improved their fraction skills with FITS; this was obviously later confirmed by their post-test scores. They also expressed the opinion that FITS made them more confident at adding fractions: in fact, Student 3 remarked that she preferred working with the computer than with her elder brother who "nagged" every time she did not understand something. They were all pleased with the way FITS praised them for doing something right or encouraged them otherwise. FITS, they therefore thought, was motivating enough (this is also a feature of the approach). Also, they were all very excited by the fact that FITS could also solve the problems it generated, and spent significant times dreaming up "difficult" problems which "even my teacher cannot do" to present to it. They were visibly happy with the explained step by step solutions to these "difficult" problems. They also liked the idea that they could enter the answer to the generated question straightaway if they knew it without being constrained to go through the various steps. In addition, two of them (Students 3 and 4) were amazed by the accuracy of its diagnoses of some of their errors. These same two students, who got the easier questions in the pre-test correct but got the harder ones wrong, remarked (to the author's greatest delight) that FITS had made them realise that they could solve the harder problems by just applying a few operations which then reduced the problem into a simpler one which they could easily manage; this was undoubtedly the most appreciated of all the comments made. These same two also liked the way FITS decided and retaught them some subskills that they were having problems with, and also how it sometimes 'led them by the hand' when they were fumbling (i.e. the socratic style). After such occasions, they appeared to get on better with the tutor. They also liked FITS's suggestions and diagnostics when they got stuck or confused.

Nevertheless, there were also aspects about FITS that the children clearly did not like, mostly to do with FITS's interface. They were all visibly fed up with the full stops they had to put after every input to this Prolog-based ITS though they appeared to get more used to it halfway through the session. Also, text on the SUN screens does not scroll gracefully and naturally; rather it tends to 'jump'. They found this very off-putting. They also did not understand some of FITS's diagnostics or texts and took great delight in suggesting better alternatives, e.g. they were all unhappy, at least initially, with the piece of text 'Please perform operation' and largely suggested 'Enter answer' as an alternative. These as well as other comments made it evident that FITS would have been a much better ITS, at least to these children, had it been developed by their expert fractions teacher since, e.g. he/she would be more versed with how best to communicate the knowledge to them. The modelling was also shown to be slightly inadequate: Students 2 & 4 were visibly annoyed by the fact that FITS sometimes retaught them certain skills which they were confident they understood.

In summary, they enjoyed using FITS and learnt from it (a collection of some of their comments is given in Appendix H). They also generally expressed the view that FITS, in its present form, could definitely be used to some significant effect at schools with children of the age range 9-14 years. However, the main conclusion of this evaluation seems to be that FITS has achieved what it was at least designed to be - a good tool to be used alongside a human tutor.

## 8.5 Evaluation of FITS's Behaviour

Self's (1985b) set of questions as regards the behaviour of an ITS were derived from an informal set of 'motherhood' axioms that should apply to any tutoring situation, computer-based or otherwise (Ford, 1988):

- The tutor should know something about the subject.
- The tutor should know something about the student.
- The student should be 'actively engaged' and not simply told.
- Learning is likely to be handicapped if the tutor and student have difficulty communicating with one another.

The fifteen questions fall into four categories relating to the above axioms:

- Subject knowledge
- Student knowledge
- Student control
- Mode of communication

This section's appraisal is concerned with whether FITS can behave in the way suggested by the questions.

### 8.5.1 Subject Knowledge

1. *Can the system answer arbitrary questions from the user about the subject?*

FITS's menu-based approach does not allow for arbitrary questions from the student about addition of fractions *per se*. This is partly due to the very restrictive interface but mostly due to the fact that the student never really has the initiative; FITS is always in control. Furthermore, this question is perhaps more relevant to discourse-type ITSs. Nevertheless, one can envisage FITS in a more mixed-initiative setting in which case the question would be very relevant, naturally with the incorporation of natural language capabilities. However FITS, from time to time, provides opportunities for questions, e.g. it would provide opportunities for a student to enter any fraction addition problem for it to solve.

2. *Can the system give an explanation of a problem solution (including a problem posed by the user)?*

A typical problem solution as seen earlier, irrespective of whether it is posed by the user or the system looks as follows:

```
Solving problem ->                              3:3/4 + 1:2/4
Adding whole numbers ->                         4 + 3/4 + 2/4
Adding equivalent fractions ->                  4 + 5/4
Change improper fraction to mixed number->      4 + 1 + 1/4
Adding whole numbers (Answer)->                 5:1/4
```

Clearly, the problem solution is self-explanatory: it explains which concepts/skills have been used at each stage in the solution of the problem. All the children who have used FITS expressed their approval of this feature.

3. *Can the system give alternative explanations, using perhaps analogy?*

Perhaps this question is also not as relevant as it would be to for systems involved with discourse domains e.g. geography. Anyway, FITS does not provide any form of alternative explanations (e.g. during preteaching) mainly due to the fact that only one view of the subject has been represented in the syllabus. It is possible to adapt FITS to support alternative explanations by say mapping a semantic net onto the syllabus which allows for this.

4. *Can the system answer hypothetical questions, that is, questions not about the present situation but about some imagined situation relating to it?*

   FITS's domain and range of expertise (addition of two numerical fractions) is so limited that the sort of hypothetical questions suggested by the question are virtually non-existent. In a discourse domain, this is a very relevant question.

### 8.5.2 Student Knowledge

5. *Could the system give a report on the student's level of understanding?*

   The student is assessed and graded by FITS (as master, novice or 'lacker' (i.e. student lacks the skill from his/her repertoire)) for each concept/task in the syllabus as well as for the overall skill of addition of fractions. It also assesses the student's knowledge of *when* to use these concepts/skills. All this information is contained in the student's model. Therefore, generating a student profile as suggested by the question from such an explicit model is trivial; it is generated on request.

6. *Are the system explanations tailored to the user?*

   This question probes whether the explanations generated during interaction with FITS are pitched at the right level for the particular student. FITS does not tailor explanations for a particular student in the way envisaged by the question and explored by Sleeman (1985c): its explanations are all pre-stored. Nonetheless, only relevant stored explanations are presented in the appropriate situations, e.g. when diagnosing a particular common misconception which a student exhibits. However, the children who tested FITS were happy with the explanations though they noted that some of them could be better phrased.

7. *Does the system provide informative feedback?*

This relates to the ability of the system to detect the need for informative feedback and to provide it if there is some evidence of misunderstanding. FITS achieves this as demonstrated by the various protocols observed so far in Sections 8.2 and 8.3 (e.g. see Figure 8.1). The children also largely expressed their approval of FITS's feedback (see Appendix H).

8. *Are the problems presented by the student adapted to the user's needs?*

FITS also achieves this with probably even more success. Pre-modelling as well as on-going modelling make this possible. Hence, concepts believed to be lacking are re-presented, whereas those only believed to be novicely mastered are reinforced by providing adequate example problems generated by FITS. Later, tutoring is also broken into various sessions of increasing difficulty levels. Therefore, unless a particular student's model reveals belief of the required standard of knowledge of that particular difficulty level, no problems of higher difficulty are presented. FITS uses the Student-History data module to prevent repeated presentations of the same task, concept or problem.

### 8.5.3 Student Control

9. *Does the system actively engage the user?*

One would guess that some students would feel FITS achieves this and others that FITS did not. Perhaps the reader should make a judgement from the protocols observed so far. FITS attempts to engage the user by always involving him/her in some interaction at most times. Perhaps, at some stages, the student would be bored by being limited to just 'Press the <RETURN> key' responses, as one of the students who used the system was. In general, FITS is quite interactive but perhaps its dominance of the interaction could frustrate some students. However, the students who have used it appeared actively engaged for most of the time when they were using the system. Admittedly, they were occasionally visibly irritated and distracted by the 'jumping' screen as previously mentioned.

10. *Can the user initiate some new area of investigation?*

FITS dominates the interaction: it decides on what to present, when to present and how to present it. FITS definitely, as explained earlier, lacks on mixed-initiative interaction which the question suggests.

11. *Does the system monitor such proposed changes, and comment on them if they seem to be unwise?*

This question follows from the former. Clearly, the response is obvious when the answer to the previous question is taken into consideration.

12. *Does the system intervene when the user appears to be having difficulty?*

A main tenet of the iterative-style approach is to provide help whenever the student is having difficulty. On this account, and as demonstrated in many protocols observed thus far, FITS achieves this very well. In fact, perhaps it should allow for more floundering in certain situations with certain students. However, the students who used it were contented with the help it provided when they were in difficulty: in fact, a couple expressed delight with some of the remedial interventions.

### 8.5.4 Mode of Communication

13. *Can the user express his inputs to the system in whatever way is most natural?*

Once more, the domain FITS tutors is so limited, and hence its inputs, that FITS achieves this with much success (this question is much more relevant for discourse domains). For example, most of the FITS's inputs are fraction expressions and it allows for all the various fractional idiosyncratic inputs most currently used, as discussed in the previous section. However, these inputs must abide by FITS's syntax of fractional expressions. On other occasions mainly integer menu entries are expected; the student really has no more 'natural' ways of expressing these. Surprisingly, the students got accustomed to FITS in about 10-15 minutes and henceforth, used it quite 'naturally'.

14. *Does the system help if the user's input is not understandable to the system?*

This is again much harder to achieve for discourse-type domains. In FITS, the most complicated inputs expected are fractional expressions. It will report a syntax error if the student's input is not parsable; it does not indicate where the parse fails. However, the grammar is so limited that one does not envisage any student's progress with the system being hampered because of incomprehensible inputs. The students usually got it right on the second trial since FITS's inputs are quite simple in nature.

15. *Are the system's outputs natural?*

All of FITS's outputs are produced from pre-stored texts and templates. This becomes evident after using the system for a reasonable length of time and it is unquestionable that it lacks the naturalness suggested by the question. Perhaps better wording of current texts and templates would enhance this aspect of its behaviour: the students actually noted this. Sleeman's (1985c) exploration into this issue and the general problem of tailoring the explanation to suit the particular user promises better success.

In summary, one concludes from the above responses that FITS rates fairly well against these set of questions: hence, it largely deserves the prefix 'intelligent', at least on the basis of Self's criteria. However, there are still several behavioural features which it lacks.

## 8.6 Evaluation of FITS's Design

O'Shea *et al.* (1984) present the thirteen "pillars" of their design philosophy for any computer-based tutor. The aim of this section is to evaluate FITS's design against these rather general principles of computer tutor construction; in turn, this reveals those aspects of FITS's design which provide scope for improvement in order to make it more useful as a computer tutor, at least according to O'Shea *et al.* (1984). Each of this "pillars" is examined in turn and a brief discussion of how successfully FITS achieves it is provided.

1. *Robustness*: FITS is largely robust. Input errors (e.g. typing mistakes) made by students which are thought by human observers to be 'obvious' mistakes are detectable by the system and reported. It is however not always specific enough as to what the exact error is.
2. *Helpfulness*: FITS is quite helpful: it always provides some sort of help when the student gets 'stuck'.
3. *Simplicity*: FITS is reasonably simple as it minimises on the amount of typing necessary to achieve a given task as observed in the various protocols so far. However, it is not as simple as systems like ALGEBRALAND which actually perform the arithmetical operations for the users. This again brings up the 'training wheels' problem. It was a dual objective in FITS to reduce this latter problem while maintaining simplicity. The students who have used FITS found it fairly easy to use.
4. *Perspicuity*: FITS is fully perspicuous: it does not provide the student with a mystifying choice from an array of several hundred buttons.
5. *Power*: FITS is totally lacking as regards power: state-of-the-art graphic capabilities are not available (at any level) to the student.
6. *Navigability*: FITS is quite navigable: the student can mostly recognise 'where he/she is' when using the system.

7. *Consistency*: FITS is absolutely consistent: it works in a clear and consistent fashion. It would behave in the same way in the same situations.
8. *Transparency*: FITS is fully transparent: the effects of the student's action are always displayed.
9. *Flexibility*: FITS is to a large extent flexible: 'wizards' or experienced student users progressed through the system faster i.e. students can deviate from over-used and long pathways and use shorter ones, e.g. providing a final answer directly or an intermediate answer closer to the final answer as opposed to going through all the stages. All the students remarked that they found it flexible in this respect.
10. *Redundancy*: FITS lacks on redundancy as it represents only a single view of the subject material. Hence, students who do not like or understand how it teaches certain concepts/tasks are sometimes frustrated with the system.
11. *Sensitivity*: FITS is to a large degree sensitive: it does tailor its tutoring according to the needs of the student. However, its explanations are not tailored as such to the particular student since they are all pre-stored.
12. *Omniscience*: FITS is quite omniscient: it is capable of leading the student 'by the hand' by asking leading questions in cases where it knows what the student wants to do but is having difficulty with doing it (see Figure 8.1).
13. *Docility*: FITS is not very docile: it is not usually seen to be under the student's command. This is due to its dominance of the interaction and sometimes the author got the feeling that it was 'steamrolling' some of the students who were using it.

## 8.7 Limitations of FITS

Naturally, there is no system without limitations; any evaluation would remain incomplete without them. There seem to be two categories of shortcomings for any such system: shortcomings due to the system's failure in achieving its specification, and retrospective limitations of the system's design. One concludes from Section 8.3 that FITS has mostly achieved its specification of Chapter 5. FITS was next appraised using other methods in Sections 8.4, 8.5 and 8.6; generally, it also appears to have fared largely well against these. However, these latter appraisals clearly reveal some behavioural and design deficiencies which include:

- FITS's explanations are not tailored to the particular student.
- FITS's outputs are not natural.
- FITS tends to have the initiative most of the time.
- FITS lacks state-of-the-art graphic capabilities.
- FITS lacks on redundancy.
- FITS does not learn or improve its strategies over time.
- FITS's student modelling is inadequate.

All current tutors (e.g. the GEOMETRY tutor) also suffer from such shortcomings as listed above. However, there is one main feature which some of these tutors provide which FITS does not match up to. Some tutors provide much better interfaces: for example, the GEOMETRY tutor generates a proof tree which reifies mental processes (see Section 3.3.1). FITS could do with such an interface without necessarily increasing the 'training wheels' problem. In fact, the students, though largely contented with the present interface, suggested that it could further be improved to make FITS more attractive.

## 8.8 Summary

This chapter has provided a reasonably comprehensive evaluation of FITS. It has appraised FITS against its specification, against some independent set of questions to determine how well it lives up to the prefix 'intelligent', and also against real students. It also presented an evaluation of the system's design along with some of its shortcomings. The next chapter provides the conclusions of this research endeavour and some suggestions for further work.

# Chapter 9

# Conclusions and Further Work

## 9.1 Introduction to Conclusions and Further Work

It is often the case that the results and lessons to be learnt from a piece of research remain 'blurred' in chapters of theses and even in a good proportion of scientific papers. This said, it is therefore necessary to provide an account which attempts to sum up the entire research reported in this thesis. This may help other researchers who might wish to further the work by helping to avoid duplication. Furthermore, suggestions for further research may enhance research continuity. The intention of this chapter is thus twofold: to set out more sharply the conclusions drawn from this research endeavour and to provide some suggestions as to how to further it.

## 9.2 Conclusions

In the course of this research, several issues have been touched on from which many conclusions can be drawn. Perhaps it is more appropriate to present the main conclusions drawn first, and next provide some other conclusions derived.

### 9.2.1 Main Conclusions

It seems reasonable to restate the main objectives of the research prior to providing any meaningful conclusions. The aims of the research were twofold (see Chapter 5):

1. To investigate an alternative approach for constructing ITSs in mathematics which improves on some of the shortcomings of existing approaches.
2. To develop and test an experimental system based on this approach for the domain of addition of fractions.

Generally, both aims were achieved with differing degrees of success: an alternative approach, the iterative-style approach, has been proposed and a system (FITS) has been constructed, which has been shown to be functional: hence demonstrating the viability of the approach. However, the main conclusions drawn from research are:

1. A model-tracing (monitoring) based approach to constructing ITSs in mathematics is a more appropriate than a mal-rule based one, especially for more complex domains. The fact that FITS has been demonstrated to be a much superior tutor to BUGGY, FRACTIONLAND or any other typical mal-rule tutor clearly supports this assertion.
2. The iterative-style approach (essentially a much-improved model-tracing approach) proposed in this thesis is a better approach to constructing ITSs in mathematics than current mal-rule and model-tracing approaches. It incorporates features which have all been demonstrated through FITS to have improved on many of the shortcomings of these current approaches. Some of the novel features of the approach over existing ones include:

   - Pre-modelling of the student.
   - Preteaching of prerequisite skills.
   - Providing tutoring sequences such that Vanlehn's felicity conditions are satisfied.
   - Modelling of the student throughout all his/her interactions with the tutor.
   - Explicitly representing knowledge, e.g. of the student or of tutoring strategies.
   - Supporting improved remedial strategies.
   - Testing the student's understanding.
   - Supporting various idiosyncratic ways which the student might choose to solve the problem.

- Motivating and supporting a more flexible style of tutoring.
- Providing environments in which the interaction between them and the students is as close as possible to the reality of the day.

However, it is by no means an ideal approach (after all, there seems to be no such thing as an 'ideal' approach to most such solutions, but just a 'better' one); evaluation has exposed some of its limitations and so there is still scope for advancement.

3. FITS, a fraction tutor based on this iterative-style approach, has been developed and demonstrated to be functional. It has also been shown to have largely improved on many of the shortcomings of existing ITSs in mathematics (e.g. BUGGY or the GEOMETRY tutor); FITS has largely met its specifications (i.e. the principles of the proposed iterative-style approach). Its behaviour as measured against Self's (1985b) set of questions to determine how well it lives up to the prefix 'intelligent' was quite encouraging: admittedly, this is to some extent due to the 'structuredness' of mathematics. Its performance with real students was also very promising though, granted, it has by no means been exhaustively tested. The children who have used FITS either visibly learnt from it (as demonstrated by the pre- and post-test scores) or expressed the opinion that they had learnt something and/or improved their fraction skills with it. For example, a couple of students actually remarked that it made them realise that a harder problem could be solved by just applying a few operations which then reduced it into a simpler one which they could easily manage. They also generally thought that it would be useful to other children in the age range of 9-14 years. This is a very reassuring result considering FITS was developed to remain only a test prototype. The fact that it is even usable by children without much help, and is also potentially useful to them is even more encouraging considering that most ITSs in the literature end up being of more

use to student teachers than to students themselves (see Chapters 3 and 4). However, the various appraisals have exposed many of its shortcomings. Thus, there is still much latitude to make it not only more acceptable as a computer tutor but also more attractive.

Nonetheless, the evaluations of Chapter 8 largely reveal that FITS has improved on many of the drawbacks of existing ITSs. This is largely attributed to its unique architecture (see Figure 6.2). In Chapter 6, it was theoretically predicted that a tutor based on such an architecture would be 'better' than those based on existing designs (see Section 6.5); this is because it largely appears to subsume the previously-mentioned architetures of Section 2.4.2. Therefore, it seems fair to say that FITS has lived up to these predicted theoretical advantages. Furthermore, it is also important to highlight the clear separation of domain-dependent and domain-independent parts achieved with this design.

From conclusion 2 above and this one, it is clear that the overall objectives of this research have been very largely met although there are several behavioural and design features which the appraisals have exposed to be lacking from the iterative-style approach, and hence from FITS.

### 9.2.2 Other Conclusions

There are several other 'side' conclusions that are derived from this piece of work besides the main ones presented above. They are:

4. Such research is well worth doing. During the course of evaluating FITS, the author's view that it is not ideal for people like himself to be constructing such systems was strengthened. It clearly emerged, from the evaluation with the students, that FITS would have been a better educational tool, had it been developed by an expert fractions teacher who had been given the necessary tools and techniques. This research has been an attempt towards such a

technique (the iterative-style approach). Admittedly, an expert teacher is not expected to be able to use this technique in its present form, but perhaps he/she could be able to make use of FITS (the tool), e.g. by modifying it to improve on some of the shortcomings which the students criticised or to tutor another domain.

However, the research concluded that the reasons for the lack of more credible techniques and tools to date have been twofold: cognitive scientists have failed to come up with a universally accepted theory of instruction/learning and that there is too little research into providing such techniques, mainly as most researchers have focussed on various components of the ITS architecture (e.g. the student model). This thesis has argued that the first problem is a long-term goal, and that the second holds better promise, at least in the short-term. The apparent 'success' of FITS bears testimony to this premise. Thus, more researchers now working on various components of the ITS architecture should redirect some of their energies towards this goal. In other words, more researchers should adopt a more 'engineering (trial and error) approach' towards ITS research, rather than the more rigorous scientific style. After all, Wernher von Braun is reputed to have said that "research is what I am doing when I don't know what I am doing". In brief, it is well worth pursuing such research.

5. Systems such as FITS have been developed mainly using the tutoring approach to using computers in mathematical education (see Chapter 1). It became evident, again from its evaluation against students, that FITS would not suffice on its own to teach addition of fractions to students with absolutely no knowledge of it, despite the fact that it has demonstrated the potential of being very useful to children with some knowledge, albeit limited. This is because FITS largely teaches the 'mechanics' of fractions addition as well as various fractions subskills, and also imparts some understanding of when to use the

various fraction skills; however, it does not convey a good understanding of the underlying concept of fractions. Though, FITS in itself surpasses most of what current mathematical ITSs can do, it would have been a much better tutor had this been achieved; the pilot studies pointed out the fact that many of the children's errors stemmed from the fundamental lack of understanding of the basic underlying concepts of fractions (i.e. they did not conceptualise it). It appears that ITSs will not be capable of conveying such 'understanding' without e.g. incorporating some of the ideas from the LOGO camp which are claimed to encourage more exploration and appreciation of mathematics, and hence more understanding. Any intelligent tutor should, ideally, be able to impart a good understanding of what it tutors. In brief, some cross-fertilisation of ideas from the tutorial and LOGO views of using computers in mathematical education promises to provide more truly intelligent tutors.

6. Current theories of bug causation (e.g. Brown & Burton's Repair Theory) are at too low a level to make any significant impact on the construction of ITSs. Such theories would (if they exist) have great implications for remedial actions as well as modifying pedagogic strategies and generally devising learning environments which pre-empt these bugs from occurring in the first place. Regrettably, existing theories appear to take a very narrow and superficial view of how bugs arise in children's computations. It is becoming more evident that mal-rules' genesis is deep rooted in the learning process. This is supported by the fact that most errors observed in the pilot study reported in Chapter 4 seem to stem from a distinct lack of *understanding* of the required concepts. In short, these theories will have to go a lot 'deeper' to have any significant impact on ITS construction. Vanlehn's (1987) STEP theory thus appears to be a move in the right direction.

## 9.3 Future Work

ITS research is generally still in its infancy and so there is still much work to be done; the research reported in this thesis has identified just a few. The main products of this work are the iterative-style approach and FITS, both of which have been noted to have several limitations. This section mostly identifies the future work to be done in order to address these shortcomings. However, some other areas also touched on in this work which require further research are also suggested. Future work which can be pursued includes:

1. FITS's explanations are not tailored to the particular student, i.e. it does not provide appropriate explanations according to the student model. Some researchers, e.g. Sleeman (1985c), have begun addressing this problem but much more still needs to be done before it warrants incorporation into ITSs.
2. FITS's outputs are not natural. Once more, this is mainly because of its lack of a capability of generating appropriate and natural explanations from student models. It is clear from FITS that prestored texts and templates are far from adequate. Perhaps this problem should be researched in tandem with 1 above.
3. FITS dominates the initiative. Much more research needs to be done towards providing more truly mixed-initiative tutors. It appears that not much progress has been made since Carbonell began addressing this problem with SCHOLAR in the early 1970s. However, some researchers, e.g. Davies *et al.* (1985), are making some progress, albeit limited.
4. FITS does not learn or improve its strategies over time. More research on self-improving tutors of the type reported in O'Shea (1982a) still needs to be pursued. In fact, some AI researchers argue that no system can claim the prefix 'intelligent' without some sort of learning capability.
5. FITS lacks state-of-the-art graphics. However, it is not trivial to incorporate such graphics into ITSs to optimise learning. There are still many cognitive factors concerned with constructing such interfaces that need to be researched

into. Fortunately, these are the central concerns of those in the Human-Computer Interaction (HCI) fraternity.

6. FITS's student modelling was shown to be inadequate during its evaluation with students. This is hardly surprising as student modelling is the key issue in ITSs and probably the most difficult to tackle. However, researchers like Self seem to be succeeding in ensuring that it receives adequate attention. Perhaps some of the research in qualitative reasoning/modelling needs to be consulted: Clancey (1988) has already begun such an investigation.
7. FITS represents only a single view of its knowledge, i.e. it lacks redundancy. Humans have the incredible ability of being able to present the same material in radically different ways; ITSs would be enhanced significantly by possessing such an ability but one suspects that this is an extremely difficult and long-term problem as humans tend to draw from their 'worldly' knowledge acquired through life's experience when doing this. Nevertheless, it is an area that needs much more looking into.
8. The ITS and LOGO fraternities both need each other in order to produce better computer-based tutors as earlier indicated. The LOGO group could contribute their rich experiences of providing some invaluable educational support for enhancing real understanding since their tutors support discovery-type learning. The ITS group has experience in implementing supportive environments for various problem solving domains. Hence, both groups could complement each other: in fact, present classroom instruction seem to have been very successfully using views similar to some of those from both fraternities. In short, there should be a 'marriage' of ideas from these two fraternities which, perhaps unfortunately, are presently competing with each other. Sleeman (1985b) also echoes such sentiments. In fact, some researchers, e.g. O'Shea *et al.* (1988), have already begun implementing this suggestion.

9. The research reported in this thesis touched on various bug theoretical aspects. It was concluded that current theories of bug causation are too inadequate to make much impact on the construction of ITSs; there is still much scope for future work. New theories of bugs, of any type, are much needed and can constitute an important contribution to the field. Further research is needed both at the empirical (or practical) level of collecting, classifying and trying to account for bugs similar to that reported in this thesis, and also at the more theoretical level of trying to understand the fundamental cognitive processes and essential ingredients underlying human knowledge acquisition and learning in general (e.g. Vanlehn's (1987) STEP theory). Such investigations would likely lead to better learning/mislearning theories which in turn should result in widely applicable contributions to the construction of ITSs.

## 9.4 Concluding Viewpoints

It is appropriate to conclude this thesis with some viewpoints on some of the contentious issues in the intelligent tutoring domain; it also has its own on-going debates. Two pertinent ones are:

1. Is intelligent tutoring just old wine in a new bottle, or is it a new vintage (Ok-choon *et al.*, 1987)?
2. Is intelligent tutoring really possible (Ridgway, 1988)?

Prior to providing responses to these questions, it seems fair to remark that they arise in the first place because of public opinion about the AI enterprise in general. Bobrow *et al.* (1986) correctly point out that public opinion about AI is schizophrenic, ranging from "it will never work" to "it might cost me my job". This range of opinion reflects the collective confusion about AI. It appears that the AI fraternity has got only itself to blame. Misleading publicity, mostly in order to attract grants, and misuse of flamboyant expressions like artificial intelligence, expert systems, intelligent tutoring systems, etc. have helped contribute to unrealistic expectations of the state-of-the-art. Consequently, questions like the two above are bound to be asked.

The first question probes if intelligent tutoring is just old wine in a new bottle, or if it is really a new vintage. The old wine was basically the AFO-type CAI which has been observed to have taken a very naive view of the instructional process. It has since been realised that providing a truly 'intelligent' tutor is a non-trivial task requiring experts from several disciplines. ITSs combine at least some of the following: AI, psychological models of the student and expert, and educational theory. There is thus a substantial change in the quality of wine and, therefore, a new vintage.

The second question investigates whether intelligent tutoring is really possible. Many answers to such questions in the past have turned out to be far too optimistic. For example, Chapter 1 notes Suppes' (1966) prediction that "in a few more years millions of school children will have access to what Phillip of Macedon's son had as royal prerogative: the personal services of a tutor as well-informed as Aristotle". Clearly, besides the obvious financial constraints forbidding this and without much deliberation on what Suppes meant by 'few', this prediction is still far from being achieved a quarter of a century on. Hence, the response to this second question requires more caution. The answer seems to depend on how the questioner views the use of the word 'intelligent' in AI terms. If he/she rightly views it as strictly speaking a misnomer, at least for now, then intelligent tutoring becomes very feasible, at least in various limited domains. On the other hand, if the questioner is more literal in his/her view of word, as many AI sceptics in the literature seem to be (again probably due to the misleading publicity), then of course intelligent tutoring would appear impossible, and so would AI in general. Nevertheless, there is a suspicion that no matter how 'intelligently' an ITS may eventually perform (e.g. even if one were to demonstrably perform better that most human tutors), these sceptics would still not be satisfied mainly, because ITSs are computer-based. It therefore appears that researchers will eventually have to turn to the famous Turing test of intelligence for an answer when ITSs come of age. This is because Turing's test circumvents the problem of lack of consensus on the definition of the word 'intelligent': it regards intelligence as undefinable but 'intelligent behaviour' as recognisable.

However, one prediction looks secure: that despite these controversies, ITS research will grow. This is because, apart from their practical needs, the area appears to provide an excellent test-bed for theories from AI scientists, educational theorists and cognitive psychologists (For instance, it has been noted that the famous Carnegie Mellon psychologist John Anderson came into the area to test his psychological theories). Furthermore, ITS researchers themselves would be very interested in incorporating any promising research results from various AI/Cognitive Science subdomains, e.g. qualitative reasoning, planning, natural language understanding, human-computer interaction etc. into their ITSs, and so they should. In conclusion, it appears certain that the best of ITS research is yet to come.

# References

Aiello, L., Carosio, M. & Micarelli, A. (1988), "An Intelligent Tutoring System for the Study of Mathematical Functions", *Proceedings of the 1st International Conference on Intelligent Tutoring Systems*, Montréal, Canada, 170-175.

Anderson, J. R. (1983), *The Architecture of Cognition*, Cambridge, MA: Harvard University Press.

Anderson, J. R. (1984), "Cognitive Psychology and Intelligent Tutoring", *Proceedings of the 6th annual conference of the Cognitive Science Society*, Boulder, CO, 37-43.

Anderson, J. R. (1987), "Production Systems, Learning and Tutoring", In Klahr, D., Langley, P. & Neches, R. (eds), *Production System Models of Learning and Development*, London: MIT Press, 437-458.

Anderson, J. R. (1988), "The Expert Module", In Polson, M. C. & Richardson, J. J. (eds), *Foundations of Intelligent Tutoring Systems*, London: Lawrence Erlbaum, 21-53.

Anderson, J. R., Boyle, C. F., Farell, R. & Reiser, B. J. (1984a), "Cognitive Principles in the Design of Computer Tutors", *Proceedings of the 6th annual conference of the Cognitive Science Society*, Boulder, CO, 2-9.

Anderson, J. R., Farell, R. & Sauers, R. (1984b), "Learning to program in LISP", *Cognitive Science* **8**, 87-129.

Anderson, J. R., Boyle, D. F. & Reiser, B. J. (1985a), "Intelligent tutoring systems", *Science* **228**, 456-462.

Anderson, J. R., Boyle, D. F. & Yost, G. (1985b), "The geometry tutor", *Proceedings of the 9th International Joint Conference on Artificial Intelligence,* Los Angeles, CA, 1-7.

Anderson, J. R. & Reiser, B. J. (1985), "The Lisp Tutor", *Byte* **10**(4).

Anderson, J. R. & Skwarecki, E. (1986), "The automated tutoring of introductory computer programming", *Communications of the ACM* **29**(9), 842-849.

Ashlock, R. B. (1976), *Error Patterns in Computation: a Semi-Programmed Approach*, London: Bell & Howell.

Attisha, M. G. (1983), "A microcomputer based tutoring system for self-improving and teaching techniques in arithmetic skills", Unpublished M.Sc. Thesis, University of Exeter.

Attisha, M. G. & Yazdani, M. (1983), "A micro-computer based tutor for teaching arithmetic skills", *Instructional Science* **12**, 333-342.

Attisha, M. G. & Yazdani, M. (1984), "An Expert System for Diagnosing Children's Multiplication Errors", *Instructional Science* **13**, 79-92.

Barr, A. & Feigenbaum, E. A. (1982), *The Handbook of Artificial Intelligence* **2**, Los Altos: Kaufmann.

Barzilay, A. (1985), "SPIRIT: A flexible Tutoring Style in an Intelligent Tutoring System", In Weisbin, R. C. (ed), *Artificial Intelligence Applications: The Engineering of Knowledge-Based Systems*, North Holland: IEE Computer Society.

Blando, J. A., Carlsen, C. & Sleeman, D. H. (1986), "Using an Intelligent Tutoring System to uncover errors of precedence in arithmetic", Paper presented at AERA, San Francisco.

Bloom, B. S. (1984), "The 2 Sigma Problem: The search for methods of group instruction as effective as one-to-one tutoring", *Educational Researcher* **13**(3), 4-16.

Bobrow, D. G., Mittal, S. & Steffik, M. (1986), "Expert systems: perils and promise", *Communications of the ACM* **29**(9), 880-893.

Boden, M. A. (1983), "Educational Implications of Artificial Intelligence", In Maxwell, W. (ed), *Thinking, The Expanding Frontier*, Philadephia: Franklin Institute Press.

Bonnet, A. (1985), *Artificial Intelligence: Promise and Performance*, London, Prentice Hall.

Borasi, R. (1986), "On the educational role of errors in mathematics: Beyond diagnosis and remediation", Unpublished doctoral dissertation, State University of New York at Buffalo.

Bratko, I. (1986), *Prolog Programming for Artificial Intelligence*, Reading, MA: Addison-Wesley.

Brown, J. S. (1983), "Learning-by-doing revisited for electronic learning environments", In White, M. A. (ed), *The Future of Electronic Learning*, London: Lawrence Erlbaum.

Brown, J. S. (1985), "Process versus product: A perspective on tools for communal and informal electronic learning", *Journal of Educational Computing Research* **1**(2), 179-201.

Brown, J. S. & Burton, R. R. (1975), "Multiple Representations of Knowledge for Tutorial Reasoning", In Bobrow, D. G. & Collins, A. (eds), *Representation and Understanding: Studies in Cognitive Science*, London: Academic Press.

Brown, J. S. & Burton, R. R. (1978), "Diagnostic models for procedural bugs in basic mathematical skills", *Cognitive Science* **2**, 155-192.

Brown, J. S. & Burton, R. R. & Bell, A. G. (1975), "SOPHIE: a step towards a reactive learning environment", *International Journal of Man-Machine Studies* 7, 675-696.

Brown, J. S. & Burton, R. R. & de Kleer, J. (1982), Pedagogical, natural language, and knowledge engineering techniques in SOPHIE I, II, and III", In Sleeman, D. H. & Brown, J. S. (eds), *Intelligent Tutoring Systems*, London: Academic Press, 227-282.

Brown, J. S., Burton, R. R. & Larkin, K. M. (1977), "Representing and Using Procedural bugs for Educational Purposes", *Proceedings of the ACM77 National Conference*, 247-255.

Brown, J. S. & Vanlehn, K. (1980), "Repair Theory : A Generative Theory of Bugs in Procedural Skills", *Cognitive Science* **2**, 379-426.

Brown, J. S. & Vanlehn, K. (1982), "Towards a Generative Theory of 'bugs'", In Carpenter, T. P., Moser, J. M. & Romberg, T. A. (eds), *Addition and Subtraction: A Cognitive Perspective*, Lawrence Erlbaum , 117-135.

Burnham, W. D. & Hall, A. R. (1985), *Prolog Programming and Applications*, London: Macmillan.

Burns, H. L. & Capps, C. G. (1988), "Foundations of Intelligent Tutoring Systems: An Introduction", In Polson, M. C. & Richardson, J. J. (eds), *Foundations of Intelligent Tutoring Systems*, London: Lawrence Erlbaum, 1-19.

Burton, R. (1982), "Diagnosing bugs in a simple procedural skill", In Sleeman, D. H. & Brown, J. S. (eds), *Intelligent Tutoring Systems*, London: Academic Press, 157-183.

Burton, R. (1988), "The Environment Module of Intelligent Tutoring Systems", In Polson, M. C. & Richardson, J. J. (eds), *Foundations of Intelligent Tutoring Systems*, London: Lawrence Erlbaum, 109-142.

Burton, R. & Brown, J.S. (1977), " A Tutoring and Student Modelling paradigm for gaming environments", *SIGCSE Bulletin* **8**, 236-246.

Burton, R. & Brown, J.S. (1982), "An investigation of computer coaching for informal learning activities", In Sleeman, D. H. & Brown, J. S. (eds), *Intelligent Tutoring Systems*, London: Academic Press, 79-98.

Cambridge Institute of Education, (1985), "New Perspectives on Mathematics Curriculum. An independent appraisal of the outcomes of APU Mathematics testing 1978-1982".

Carbonell, J. R. (1970), "AI in CAI: an artificial intelligence approach to computer-assisted instruction", *IEEE Transactions on Man-Machine Systems* **11**, 190-202.

Carbonell, J. R. (1971), "Artificial Intelligence and Large Interactive Man Computer Systems", *Proceedings of the Joint National Conference on Major Systems*", Anahein, CA, 167-173.

Carbonell, J., Michalski, R. & Mitchell, T. (1983), "An Overview of Machine Learning", In Michalski, R. S., Carbonell, J. G. & Mitchell, T. M. (eds), *Machine Learning*, Palo Alto: Tioga, 3-23.

Carpenter, T. D., Coburn, T. G., Reys, R. E. & Wilson, J. W. (1976), "Notes from National Assessment: addition and multiplication of fractions", *The Arithmetic Teacher* **23**, 137-141.

Cawsey, A. (1987), "Bugs in Decimal Addition: Model, Applications and Explanations", *AISB Quarterly* **59**, 12-13.

Chan, T. W. & Baskin, B. A. (1988), "Studying with the Prince - The Computer as a Learning Companion", *Proceedings of the 1st International Conference on Intelligent Tutoring Systems*, Montréal, Canada, 194-200.

Clancey, W. J. (1982), "Tutoring rules for guiding a case method dialogue", In Sleeman, D. H. & Brown, J. S. (eds), *Intelligent Tutoring Systems*, London: Academic Press, 201-225.

Clancey, W. J. (1983), "GUIDON", *Journal of Computer-Based Instruction* **10**(1&2), 8-15.

Clancey, W. J. (1984), "Methodology for building an intelligent tutoring system", In Kintsch, W., Miller, J. R., & Polson, P. G. (eds), *Methods and Tactics in Cognitive Science*, London: Lawrence Erlbaum.

Clancey, W. J. (1987), *Knowledge-Based Tutoring*, London: MIT Press.

Clancey, W. J. (1988), "The role of qualitative models in instruction", In Self, J. A. (ed), *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction*, London: Chapman & Hall, 49-68.

Clancey, W. J & Letsinger, R. (1981), "NEOMYCIN: reconfiguring a rule-based expert system for application to teaching", *Proceedings of the 7th International Joint Conference on Artificial Intelligence* , Vancouver, Canada, 829-835.

Collins, A. & Brown, J. S. (1988), "The Computer as a Tool for learning through Reflection", In Mandl, H. & Lesgold, A. (eds), *Learning Issues for Intelligent Tutoring Systems*, London: Springer-Verlag, 1-18.

Cox, L. S. (1975), "Diagnosing and Remediating Systematic Errors in Addition and Subtraction Computations", *Arithmetic Teacher* 22, 151-157.

Crowder, N. A. (1959), "Automatic tutoring by means of intrinsic programming", In Galanter, E. (ed), *Automatic Teaching: The State of the Art*, New York: Wiley, 109-116.

Dalenoort, G. J. (1988), Personal communication.

Davies, N. G., Dickens, S. L. & Ford, L. (1985), "Tutor - A Prototype ICAI System", In Bramer, M. A. (ed), *Research and Development in Expert Systems*, Cambridge, Cambridge University Press.

Dede, C. (1986), "A Review and Synthesis of recent research in computer-assisted instruction", *International Journal of Man-Machine Studies* 24, 329-353.

De Gery, C.J., Drouard, J., Dumont, B., Hocquenghem, S., Lacombe, D. & Sol, G. (1985), "Computer Assisted Testing by Questions and Answers", In Duncan, K. & Harris, D. (eds), *Computers in Education*, North Holland: Elsevier Science, 311-316.

Downes, L. W. and Paling, D. (1958), *The Teaching of Arithmetic in Primary Schools*, Oxford University Press.

Du Boulay, J. B. (1978), "Learning primary mathematics through computer programming", Ph.D. Thesis, University of Edinburgh.

Duchastel, P. (1986), "Intelligent Computer Assisted Instruction Systems: The Nature of Learner Control", *Journal of Educational Computing Research* 2(3), 379-393.

Dumont, B. (1988), "Fraction Erreurs et Intelligence Artificielle", Thèse doctorat d'état, Université de Paris 7, France.

Eisenberg, T. A. (1976), "Computational errors made by teachers of arithmetic: 1930, 1973", *The Elementary School Journal*, 229-237.

Elsom-Cook, M. (1987), "Intelligent Computer-Aided Instruction research at the Open University", Technical Report No: 63, Computer-Assisted Learning Research Group, The Open University, Milton Keynes: UK.

Engelhardt, J. M. (1977), "Analysis of children's computational errors: a qualitative approach", *British Journal of Educational Psychology* **47**, 149-154.

Evertsz, R. (1982), "A production system account of children's errors in fraction subtraction", Technical Report No: 28, Computer-Assisted Learning Research Group, The Open University, Milton Keynes.

Farell, R. (1987), "Intelligent Case Selection and Presentation", *Proceedings of the 10th International Joint Conference on Artificial Intelligence* **1**, Milan, Italy, 174-176.

Farell, R., Anderson, J. R. & Reiser, B. J. (1984), "An Interactive Computer-Based Tutor for LISP", *Proceedings of the National Conference on Artificial Intelligence - AAAI 84*, Austin, Texas, 106-109.

Ford, L. (1987a), "Teaching Strategies and Tactics in intelligent computer aided instruction", *Artificial Intelligence Review* **1**, 201-215.

Ford, L. (1987b), "Anatomy of an ICAI system", In Lewis, R. & Tagg, E. D. (eds), *Trends in Computer Assisted Instruction*, London: Blackwell Scientific, 22-31.

Ford, L. (1988), "The appraisal of an ICAI system", In Self, J. A. (ed), *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction*, London: Chapman & Hall, 109-123.

Ford, L. & Yazdani, M. (1987), "Alvey-IKBS Research Workshop on Tutoring Systems: Workshop Report", Working paper no: 170, Dept. of Computer Science, University of Exeter.

Ford, L. & Yazdani, M. (1988), "Tutoring systems: the state-of-the-art in Great Britain", Expert Systems 5(4), 328-339.

Foss, C. L. (1987), "The Acquisition of Error Management Skills", *Proceedings of the 3rd International Conference on Artificial Intelligence and Education*, Pittsburgh.

Galanter, E. (ed) (1959), *Automatic Teaching*, New York: Wiley.

Genesereth, M. R. (1982), "The role of plans in intelligent teaching systems", In Sleeman, D. H. & Brown, J. S. (eds), *Intelligent Tutoring Systems*, London: Academic Press, 137-155.

Gilmore, D. & Self, J. A. (1988), "The application of machine learning to intelligent tutoring systems", In Self, J. A. (ed), *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction*, London: Chapman & Hall, 179-196.

Ginsburg, H. P. (1983), "Cognitive Diagnosis of Children's Arithmetic", *Proceedings of a joint conference in psychology*, Washington D. C., 287-300.

Goldstein, I. P. (1982), "The genetic graph: a representation for the evolution of procedural knowledge", In Sleeman, D. H. & Brown, J. S. (eds), *Intelligent Tutoring Systems*, London: Academic Press, 51-77.

Grignetti, M., Hausman, C. L. & Gould, L. (1975), "An intelligent on-line assistant and tutor: NLS-SCHOLAR", *Proceedings of the National Computer Conference*, 775-781.

Hart, K. M. (ed) (1986), *Children's Understanding of MATHEMATICS:11 ~ 16*, London: John Murray.

Hartley, J. R. & Sleeman, D. H. (1973), "Towards more intelligent teaching systems", *International Journal of Man-Machine Studies* **5**, 215-236.

Hasemann, K. (1981), "On Difficulties with Fractions", *Educational Studies in Mathematics* **12**, 71-87.

Hawkes, W. L., Sharon, J. D., Kandel, A. & Taps Project Staff (1986), "Fuzzy Expert Systems for an Intelligent Computer-Based Tutor", Technical Report No: 86-5, Learning Systems Institute, Centre for Educational Technology, Florida State University.

Heines, J. M. & O'Shea, T. (1985), "The design of a rule based CAI tutorial", *International Journal of Man-Machine Studies* **23**, 1-25.

Hershkowitz, R., Vinner, S. & Bruckheimer, M. (1980), "Some Cognitive factors as causes of mistakes in the addition of fractions", *Proceedings of the 4th International Conference for the psychology of mathematics in Education*, Berkeley, CA, 24-31.

Ikeda, M., Mizoguchi, R. & Kakusho, O. (1988), "Design of a general framework for ITS", *Proceedings of the 1st International Conference on Intelligent Tutoring Systems*, Montréal, Canada, 82-89.

Jones, M. A. & Tuggle, F. D. (1979), "Inducing explanations for errors in computer-assisted instruction", *International Journal of Man-Machine Studies* **11**, 301-324.

Kearsley, G. (ed) (1987), *Artificial Intelligence and Instruction Applications and Methods*, Reading, MA: Addison-Wesley.

Kelly, A. E., Sleeman, D. H., Ward, R. D. & Martinak, R. (1987), "TPIXIE: A computer program to teach diagnosis of algebra errors", Technical Report AUCS/TR8710, Dept. of Computing Science, University of Aberdeen.

Kerslake, D. (1986), *Children's Strategies and Errors*, Oxford: NFER-NELSON.

Kimball, R. A. (1982), "A self-improving tutor for symbolic integration", In Sleeman, D. H. & Brown, J. S. (eds), *Intelligent Tutoring Systems*, London: Academic Press, 283-307.

Kintsch, W., Miller, J. R. & Polson, P. G. (eds) (1984), *Methods and Tactics in Cognitive Science*, NJ: Lawrence Erlbaum.

Koffman, E. B. & Blount, S. E. (1975), "Artificial Intelligence and Automatic Programming in CAI", *Artificial Intelligence* **6**, 215-234.

Lankford, F.G., Jr. (1972), "Some computational strategies of seventh grade pupils", Washington, D.C., U.S. Department of Health, Education, and Welfare, Office of Education, National Centre for Educational Research and Development [Regional Research Program] and Charlottesville, Virginia: The Centre for Advanced Study, University of Virginia.

Lantz, B. S., Brogar, W. S. & Farley, A. M. (1983), "An Intelligent CAI system for teaching equation solving", *Journal of Computer-Based Instruction* **10**(1&2), 35-42.

Larkin, K. M. (1977), "Representing and using procedural bugs for educational purposes", *Proceedings of the ACM77 National Conference*, 247-255.

Lawler, R. W. & Yazdani, M. (eds) (1987), *Artificial Intelligence and Education* **1**: *Learning Environments & Tutoring Systems*, Norwood: Ablex.

Lepper, M. R. & Chabay, R. W. (1988), "Socializing the Intelligent Tutor: Bringing Empathy to Computer Tutors", In Mandl, H. & Lesgold, A. (eds), *Learning Issues for Intelligent Tutoring Systems*, London: Springer-Verlag, 242-257.

Lesgold, A. (1988), "Toward a theory of curriculum for use in designing intelligent instructional systems", In Mandl, H. & Lesgold, A. (eds), *Learning Issues for Intelligent Tutoring Systems*, London: Springer-Verlag, 114-137.

Lewis, M. W., Milson, R. & Anderson. J. R. (1987), "The TEACHER APPRENTICE: Designing an intelligent authoring system for high school mathematics", In Kearsley, G. (ed), *Artificial Intelligence and Instruction: Instruction and Methods*, Reading, MA: Addison-Wesley, 269-301.

Lovell, K. (1980), "Intelligent teaching systems and the teaching of mathematics", *AISB Quaterly* **38**, 26-30.

Lumb, D. (1974), "Mathematical competence", *Mathematics Teaching* **68**, 48-50.

Mandl, H. & Lesgold, A. (eds) (1988), *Learning Issues for Intelligent Tutoring Systems*, London: Springer-Verlag.

Marshall, S. P. (1980), "Procedural Networks and Production Systems in Adaptive Diagnosis", *Instructional Science* **9**, 129-143.

Martinak, R., Sleeman, D. H., Kelly, A. E., Moore, J. & Ward, R. (1987), "Studies of Diagnosis and Remediation with High School Algebra Students", Technical Report AUCS/TR8711, Dept. of Computing Science, University of Aberdeen.

Matz, M. (1982), "Towards a process model for high school algebra errors", In Sleeman, D. H. & Brown, J. S. (eds), *Intelligent Tutoring Systems*, London: Academic Press, 25-50.

Michalski, R. S., Carbonell, J. G. & Mitchell, T. M. (eds) (1983), *Machine Learning*, Palo Alto: Tioga.

Minsky, M. L. (1972), *Computation: Finite and Infinite Machines*, London: Prentice-Hall.

Moore, J. & Sleeman, D. H. (1987), "Enhancing PIXIE's Tutoring Capabilities", Technical Report AUCS/TR8709, Dept. of Computing Science, University of Aberdeen.

Nwana, H. S. (1986), "Microcomputer Based Tutors for Diagnosing Addition and Subtraction Errors", Unpublished M.Sc. Thesis, Aston University.

Ohlsson, S. (1986), "Some principles of intelligent tutoring", *Instructional Science* **14**, 293-326.

Ohlsson, S. (1987), "A semantics for fraction concepts", Unpublished Technical Report, Centre for Study of Learning Research and Development Centre, University of Pittsburgh.

Ok-choon, P., Ray, S. P. & Seidel, R. J. (1987), "Intelligent CAI: Old wine in new bottles or a new vintage?", In Kearsley, G. (ed), *Artificial Intelligence and Instruction: Instruction and Methods*, Reading, MA: Addison-Wesley, 11-43.

Oliver, W. P. (1973), "Computer-assisted mathematics instruction for community college students", *International Journal of Man-Machine Studies* **5**, 385-395.

O'Shea, T. (1979), "Rule-based computer tutors", In Michie, D. (ed), *Expert Systems in the Micro Electronic Age*, Edinburgh University Press, 226-232.

O'Shea, T. (1982a), "A self-improving quadratic tutor", In Sleeman, D. H. & Brown, J. S. (eds), *Intelligent Tutoring Systems*, London: Academic Press, 283-307.

O'Shea, T. (1982b), "Intelligent systems in education", In Michie, D. (ed), *Introductory Readings in Expert Systems*, London: Gordon & Breach, 141-176.

O'Shea, T., Bornat, R., Du Boulay, B., Eisenstadt, M. & Page, I. (1984), "Tools for creating intelligent computer tutors", In Elithorn & Beneiji, R. (eds), *Artificial and Human Intelligence*, North Holland: Elsevier, 181-199.

O'Shea, T., Evertsz, R., Hennessy, S., Floyd, A., Fox, M. & Elsom-Cook, M. (1988), "Design choices for an intelligent arithmetic tutor", In Self, J. A. (ed), *Artificial Intelligence and Human Learning: Intelligent computer-aided instruction*, London: Chapman & Hall, 257-275.

O'Shea, T. & Self, J. (1983), *Learning and Teaching with Computers*, Sussex: Harvester Press.

O'Shea, T. & Sleeman, D. H. (1973), "A design for an adaptive self-improving teaching system" In Rose, J. (ed), *Advances in Cybernetics*, London: Gordon & Breach.

Papert, S. (1980), *Mindstorms: Children, Computers and Powerful Ideas*, New York: Basic Books.

Payne, S. J. & Squibb, H. R. (1987), "Understanding algebra errors: the psychological status of mal-rules", CERCLE Technical Report No: 43, University of Lancaster.

Radatz, H. (1979), "Error Analysis in Mathematics Education", *Journal for Research in Mathematics Education*, May, 163-172.

Reiser, B. J., Anderson, J. R., and Farrel, R. G. (1985), "Dynamic Student Modelling in an Intelligent Tutor for LISP Programming. *Proceedings of the 9th International Joint Conference on Artificial Intelligence* **1**, 8-13.

Resnick, L. (1982), "Syntax and Semantics in learning to subtract", In Carpenter, T., Moser, J. & Romberg, T. (eds), *Addition & Subtraction: A Cognitive Perspective*, Hilsdale, NJ: Lawrence Erlbaum.

Resnick, L. & Ford, W. (1981), *The psychology of mathematics for instruction*, Hilsdale: Lawrence Erlbaum.

Resnick, L. B., & Gelman, R. (1983), "Mathematical and Scientific Knowledge: An Overview", *Proceedings of a Joint Conference in Psychology*, Washington D.C, 267-285.

Rich, E. (1979), "User Modelling via Stereotypes", *Cognitive Science* **3**, 329-354.

Ridgway, J. (1988), "Of course ICAI is impossible... worse though, it might be seditious", In Self, J. A. (ed), *Artificial Intelligence and Human Learning: Intelligent computer-aided instruction*, London, Chapman & Hall, 28-48.

Roberts, G. H. (1968), "The failure strategies of third grade arithmetic pupils", *The Arithmetic Teacher* **15**, 442-446.

Roecks, A. L. (1981), "How many ways can the computer be used in education? A baker's dozen", *Educational Technology* **21**(9), 16-26.

Ross, P. (1987), "Intelligent Tutoring Systems", *Journal of Computer Assisted Learning* **3**, 194-203.

Ross, P., Jones, J. & Millington, P. (1987), "User modelling in intelligent teaching and tutoring", In Lewis, R. & Tagg, E. D. (eds), *Trends in Computer Assisted Instruction*, London: Blackwell, 32-44.

Self, J. A. (1974), "Student models in computer-aided instruction", *International Journal of Man-Machine Studies* **6**, 261-276.

Self, J. A. (1985a), "A Perspective on Intelligent Computer-Assisted Learning", *Journal of Computer Assisted Learning* **1**, 159-166.

Self, J. A. (1985b), "Intelligent computer assisted instruction", Paper presented at the ICAI Spring Seminar, Logica Cambridge Ltd, Cambridge.

Self, J. A. (1987a), "The application of machine learning to student modelling", In Lawler, R. W. & Yazdani, M. (eds), *Artificial Intelligence and Education* **1**: *Learning Environments & Tutoring Systems*, Norwood: Ablex, 267-280.

Self, J. A. (1987b), "Realism in Student Modelling", *Alvey-IKBS Research Workshop Tutoring Systems*, University of Exeter.

Self, J. A. (ed) (1988a), *Artificial Intelligence and Human Learning: Intelligent computer-aided instruction*, London, Chapman & Hall.

Self, J. A. (1988b), "Bypassing the intractable problem of student modelling", *Proceedings of the 1st International Conference on Intelligent Tutoring Systems*, Montréal, Canada, 18-24.

Self, J. A. (1988c), "Student models: what use are they?", In Ercoli, P. & Lewis, R. (eds), Artificial Intelligence Tools in Education, Amsterdam: North-Holland, 73-86.

Shortliffe, E. H. (1976), *Computer Based Medical Consultations: MYCIN*, New York: Elsevier.

Skinner, B. F. (1954), "The science of learning and the art of teaching", *Harvard Educational Review* **24**, 86-97.

Skinner, B. F. (1958), "Teaching Machines", *Science* **128**, 969-977.

Sleeman, D. H. (1981), "A Rule - Based Task Generation System", *Proceedings of the 7th International Joint Conference on Artificial Intelligence* **2**, Vancouver, B.C., Canada, 882-887.

Sleeman, D. H. (1982a), "Inferring (Mal) rules from pupils' protocols", *Proceedings of the European AI Conference*, 160-164.

Sleeman, D. H. (1982b), "Assessing aspects of competence in basic algebra," In Sleeman, D. H. & Brown, J. S. (eds), *Intelligent Tutoring Systems*, London: Academic Press, 185-199.

Sleeman, D. H. (1983a), "A rule-directing modelling system", In Michalski, R., Carbonell, J. & Mitchell, T. M, (eds), *Machine Learning*, Palo Alto: Tioga, 483-510.

Sleeman, D. H. (1983b), "Intelligent Tutoring Systems: A Review", *Proceedings of EdCompCon '83 meeting*, IEEE Computer Society, 95-101.

Sleeman, D. H. (1983c), "Inferring student models for intelligent computer-aided instruction", In Michalski, R. S., Carbonell, J. G. & Mitchell, T. M. (eds), *Machine Learning*, Palo Alto, Tioga.

Sleeman, D. H. (1984a), "Mis-generalistion: an explanation for observed mal-rules", *Proceedings of the 6th annual conference of the Cognitive Science Society*, Boulder, CO, 1-6.

Sleeman, D. H. (1984b), "An attempt to understand students' understanding of basic algebra", *Cognitive Science* **6**, 127-149.

Sleeman, D. H. (1985a), "Basic algebra revisited: a study with 14-year-olds", *International Journal of Man-Machine Studies* **22**, 127-149.

Sleeman, D. H. (1985b), "AI and Education: Two ideological positions", *AISB Quaterly* **55/56**, 26-31.

Sleeman, D. H. (1985c), "UMFE: A User Modelling Front-End Subsystem", *International Journal of Man-Machine Sudies* **23**, 71-88.

Sleeman, D. H. (1987a), "PIXIE: A Shell for developing Intelligent Tutoring Systems", In Lawler, R. & Yazdani, M. (eds), *Artificial Intelligence and Education* **1**, NJ: Ablex, 239-265.

Sleeman, D. H. (1987b), "Micro-SEARCH: A shell for building systems to help students solve non-deterministic tasks", In Kearsley, G. (ed), *Artificial Intelligence and Instruction: Instruction and Methods*, Reading, MA: Addison-Wesley, 69-81.

Sleeman, D. H. (1987c), "Some challenges for intelligent tutoring systems", *Proceedings of the 10th International Joint Conference on Artificial Intelligence* **2**, Milan, Italy, 1166-1168.

Sleeman, D. H. and Brown, J. S. (eds) (1982a), *Intelligent Tutoring Systems*, London: Academic Press.

Sleeman, D. H. and Brown, J. S. (1982b), "Introduction: Intelligent Tutoring Systems", In Sleeman, D. H. & Brown, J. S. (eds), *Intelligent Tutoring Systems*, London: Academic Press, 1-11.

Sleeman, D. H. & Smith, M. J. (1981), "Modelling Students' Problem Solving", *Artificial Intelligence* **16**, 171-188.

Stevens, A., Collins, A. & Goldin, S. E. (1982), "Misconceptions in students' understanding", In Sleeman, D. H. & Brown, J. S. (eds), *Intelligent Tutoring Systems*, London: Academic Press, 13-24.

Streitz, N, A. (1988), "Mental models and metaphors: Implications for the design of adaptive user-system interfaces", In Mandl, H. & Lesgold, A. (eds), *Learning Issues for Intelligent Tutoring Systems*, London: Springer-Verlag, 164-186.

Suppes, P. (1966), "The uses of computers in education", *Scientific American* **215**, 206-221.

Suppes, P. (1967), "Some theoretical models for mathematics learning", *Journal of Research and Development in Education* **1**, 5-22.

Suppes, P. (1988), "The Future of Intelligent Tutoring Systems: Problems and Potential", *Proceedings of the 1st International Conference on Intelligent Tutoring Systems*, Montréal, Canada.

Tennyson, R. D. & Park, O-C. (1980), "The Teaching of Concepts: a review of instructional design research literature", *Review of Educational Research* **50**(1), 55-70.

Thornton, Ticker, Dossey & Bazik. (1983), *Teaching mathematics to children with special needs*, Reading, MA: Addison-Wesley.

Tobias, S. (1985), "Computer Assisted Instruction", In Wang, M. C. & Waldberg, H. J. (eds), *Adapting Instruction to Individual Differences*, Berkeley, CA: McCutchan, 135-159.

Todd, R. R. (1988), "Self-organised learning within an Intelligent Teaching Student", In Self, J. A. (ed), *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction*, London: Chapman & Hall, 197-211.

Uhr, L. (1969), "Teaching machine programs that generate problems as a function of interaction with students", *Proceedings of the 24th National Conference*, 125-134.

Van Beek, D., Van Maanen, L., Ossebaard, C., Been, P. H., Jorna, R. J. & Mulder, G. (1987), "Fractions, Pupils and their instruction in Arithmetic 1", *Cognitive Systems* **2**(1), 81 -100.

Vanlehn, K. (1982), "Bugs are not enough: empirical studies of bugs, impasses and repairs in procedural skills", *Journal of Mathematical Behaiviour* **3**, 3-72.

Vanlehn, K. (1987), "Learning one subprocedure per lesson", *Artificial Intelligence* **31**(1), 1-40.

Vanlehn, K. (1988a), "Towards a Theory of Impasse Driven Learning", In Mandl, H. & Lesgold, A. (eds), *Learning Issues for Intelligent Tutoring Systems*, London: Springer-Verlag, 19-41.

Vanlehn, K. (1988b), "Student Modelling", In Polson, M. C. & Richardson, J. J. (eds), *Foundations of Intelligent Tutoring Systems*, London: Lawrence Erlbaum, 55-78.

Visetti, Y. M. & Dague, P. (1987), "Plan inference and student modelling in ICAI", *Proceedings of the AAAI-87 6th National Conference on Artificial Intelligence* **1**, Seattle, Washington, 77-81.

Wachsmuth, I. (1988), "Modelling the Knowledge Base of Mathematical Learners: Situation-Specific and Situation-Nonspecific Knowledge", In Mandl, H. & Lesgold, A. (eds), *Learning Issues for Intelligent Tutoring Systems*, London: Springer-Verlag, 63-79.

Wenger, E. (1987), *Artificial Intelligence and Tutoring Systems*, Los Altos, CA: Morgan Kaufmann.

West, T. A. (1974), "Diagnosing Pupils' Errors: Looking for patterns", *Arithmetic Teacher* **20**, 467-469.

Wexler, J. D. (1970), "Information Networks in Generative Computer-Assisted Instruction", *IEEE Transactions on Man-Machine Systems* **11**(4), 181-190.

Woodroffe, M. R. (1988), "Plan recognition and intelligent tutoring systems", In Self, J. A. (ed), *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction*, London: Chapman & Hall, 212-225.

Woods, P. & Hartley, J.R. (1971), "Some learning models for arithmetic tasks and their use in computer-based learning", *British Journal of Educational Psychology* **41**(1), 35-48.

Woolf, B. (1987), "Theoretical frontiers in building a machine tutor", In Kearsley, G. (ed), *Artificial Intelligence and Instruction: Instruction and Methods*, Reading, MA: Addison-Wesley, 229-267.

Woolf, B. (1988a), "20 years in the trenches: what have we learned?", *Proceedings of the 1st International Conference on Intelligent Tutoring Systems*, Montréal, Canada, 33-39.

Woolf, B. (1988b), "Representing complex knowledge in an intelligent machine tutor", In Self, J. A. (ed), *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction*, London: Chapman & Hall, 3-27.

Woolf, B. & Cunningham, P. A. (1987), "Building community memory for intelligent tutoring systems", *Proceedings of the AAAI-87 6th National Conference on Artificial Intelligence* **1**, Seattle, Washington, 77-81.

Woolf, B. & McDonald, D, D. (1984), "Building a computer tutor: Design issues", *IEEE Computer*, September, 61-73.

Woolf, B., Murray, T., Suthers, D. & Schultz, K. (1988), "Knowledge Primitives for Tutoring Systems", *Proceedings of the 1st International Conference on Intelligent Tutoring Systems*, Montréal, Canada, 491-498.

Yazdani, M. (1983), "Introduction: Artificial Intelligence and Education", In Yazdani, M. (ed), *New Horizons in Educational Computing*, New York: Wiley.

Yazdani, M. (1986a), "Intelligent tutoring systems survey", *Artificial Intelligence Review* **1**, 43-52.

Yazdani, M. (ed) (1986b), *Artificial Intelligence. Principles and Applications*, London: Chapman & Hall.

Yazdani, M. (1987), "Intelligent Tutoring Systems: An Overview", In Lawler, R. & Yazdani, M. (eds), *Artificial Intelligence and Education* **1**, NJ: Ablex, 182-201.

Young, R. M & O'Shea, T. (1981), "Errors in Childrens' Subtraction", *Cognitive Science* **5**, 153-177.

Zissos, A. Y. & Witten, I. H. (1985), "User Modelling for a computer coach: a case study", *International Journal of Man-Machine Sudies* **23**, 729-750.

# Bibliography

Bundy, A., Du Boulay, B, Howe, J & Plotkin, G. (1984), "How to get a Ph.D. in AI", In O'Shea, T. & Eisenstadt, M. (eds), *Artificial Intelligence, Tools, Techniques & Applications*, New York: Harper & Row, 139-154.

Davis, R. B. (1984), *Learning Mathematics: The cognitive science approach to mathematics education*, London: Croom Helm.

Dickson, L., Brown, M. & Gibson, O. (1983), *Children learning mathematics - A teachers guide to recent research*, Holt Education.

Ford, L. (1984), "Intelligent Computer Aided Instruction", In Yazdani, M. & Narayanan, A. (eds), *Artificial Intelligence: Human Effects*, New York: Wiley.

Ginsburg, H. P. (1977), *Childrens arithmetic, the learning process*, New York: Van Nostrand.

Glaser, R. (1985), "Cognition and Adaptive Education", In Wang, M. C. & Waldberg, H. J. (eds), *Adapting Instruction to Individual Differences*, Berkeley, CA: McCutchan, 82-90.

Hartley, J. R. (1973), "The design and evaluation of an adaptive teaching system", *International Journal of Man-Machine Studies* 5, 421-436.

Hartley, J. R. (1987), "Computer-based support systems for learning and teaching" In Lewis, R. & Tagg, E. D. (eds), *Trends in Computer Assisted Instruction*, London: Blackwell, 2-15.

Langley, P. (1983), "Learning effective search strategies", *Proceedings of the 8th International Joint Conference on Artificial Intelligence* **1**, Karlsruhe, Germany, 419-421.

Lepper, M. R. & Chabay, R. W. (1985), "Intrinsic motivation and instruction: Conflicting views on the role of motivational processes in computer-based education", *Educational Psychologist* **20**, 217-230.

Littman, D. & Soloway, E. (1988), "Evaluating ITSs: The Cognitive Science Perspective", In Polson, M. C. & Richardson, J. J. (eds), *Foundations of Intelligent Tutoring Systems*, London: Lawrence Erlbaum, 209-242.

Merrill, D. M. (1985), "Where is the authoring in authoring systems?", *Journal of Computer-Based Instruction* **12**(4), 90-96.

Payne, S. J. (1988), "Methods and mental models in theories of cognitive skill", In Self, J. A. (ed), *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction*, London: Chapman & Hall, 69-87.

Reigeluth, C. M. (1983), "Meaningfulness and Instruction: Relating what is being learned to what a student knows", *Instructional Science* **12**, 197 - 218.

Reisman, F. K. (1982), *A Guide to Diagnostic Teaching of Arithmetic*, Ohio: Charles E. Merrill.

Rich, E. (1983), *Artificial Intelligence*, McGraw-Hill.

Sandberg, J. A. (1987), "The Third International Conference on Artificial Intelligence & Education", *AICOM* **0**(1), 51-53.

Self, J. A. (1986), "Artificial Intelligence - Its potential in education and training", Paper presented at the 5th Canadian Symposium on Instructional Technology, Ottawa.

Seltzer, R. A. (1971), "Computer-Assisted Instruction - what it can and cannot do", *American Psychologist Journal* **26**, 373-377.

Simon, H. A. (1983), "Why should machines learn?", In Michalski, R. S., Carbonell, J. G. & Mitchell, T. M. (eds), *Machine Learning*, Tioga, 25-37.

Suppes, P. (1979), "Current Trends in Computer Assisted Instruction", *Advances in Computers* **18**, New York: Academic Press, 173-229.

Underhill, B. (1976), *Teaching Elementary School Mathematics*, Colombus, Ohio: Charles Merrill.

Vanlehn, K. (1983), "Validating a theory of human skill acquisition", *Proceedings of International Machine Learning Workshop*, Illinois, 234-240.

Wang, M. C. & Waldberg, H. J. (eds) (1985), *Adapting Instruction to Individual Differences*, Berkeley, CA.

Woods, S. S., Resnick, B. L. & Groen, G. J. (1975), "An experimental test of five process models for subtraction", *Journal of Educational Psychology* **67**(1), 17-21.

Woolf, B. (1986), "Teaching a complex industrial process", *Proceedings of the AAAI-86 5th National Conference on Artificial Intelligence*, Philadephia, PA, 722-727.

Yazdani, M. (1984a), "AI and Education", In Yazdani, M. & Narayanan, A. (eds), *Artificial Intelligence: Human Effects*, New York: Wiley.

Yazdani, M. (ed) (1984b), *New Horizons in Educational Computing*, New York: Wiley.

Yazdani, M & Lawler, R. W. (1986), "Artificial Intelligence and Education: An Overview", *Instructional Science* **14**, 197-206.

Yazdani, M. & Narayanan, A. (eds) (1984), *Artificial Intelligence: Human Effects*, New York: Wiley.

## Appendix A

## The Pre-Test Problems

1. $2 + 3 =$
2. $2 \times 4 =$
3. $8 \div 2 =$
4. $2 \times 3 =$
5. Simplify by cancelling.

   a. $^{3}/_{3}$

   answer =

   b. $^{10}/_{12}$

   answer =

6. a. Find the smallest number into which 3 and 4 will both divide.

   b. Find the smallest number into which 4 and 6 will both divide.

7. $5 - 4 =$
8. These are inproper fractions. Change them to mixed numbers.

   a. $^{4}/_{3}$

   answer =

   b. $^{5}/_{4}$

   answer =

9. $24 \div 3 =$
10. Find the smallest number into which 4 and 8 will both divide.

## Appendix B

## Problem Set for Addition of Fractions

1. $^1/_6 + ^1/_4$
2. $^5/_7 + ^4/_7$
3. $2\,^1/_5 + 1\,^1/_5$
4. $2\,^2/_6 + 1\,^2/_4$
5. $^3/_8 + 3\,^1/_8$
6. $^1/_4 + ^2/_8$
7. $^2/_3 + 1$
8. $^3/_8 + ^2/_8$
9. $2\,^1/_8 + 1\,^1/_4$
10. $3\,^2/_3 + 1\,^2/_3$
11. $^1/_6 + 2\,^3/_8$
12. $^3/_8 + ^1/_6$
13. $2 + ^1/_4$
14. $1\,^1/_4 + ^1/_3$
15. $1\,^2/_3 + 2\,^3/_4$
16. $1\,^3/_5 + 2\,^7/_{10}$
17. $1\,^5/_6 + 1\,^1/_8$
18. $^2/_{10} + ^3/_5$
19. $1\,^1/_2 + ^3/_4$
20. $^1/_3 + ^1/_4$
21. $1\,^1/_2 + 1\,^1/_3$
22. $^2/_3 + ^3/_5$
23. $^1/_2 + 7\,^1/_3$
24. $^1/_6 + 2\,^1/_2$
25. $1\,^1/_4 + ^5/_6$
26. $4\,^2/_3 + ^1/_3$

27. Extra problem designed to uncover a particularly keen child's misconception.

$^{7}/_{20} + ^{2}/_{5}$

## Appendix C

## Problem Set for Subtraction of Fractions

1. $^5/_6$ - $^1/_3$
2. 2 $^5/_8$ - $^4/_5$
3. 7 $^2/_3$ - 2 $^2/_3$
4. 1 $^5/_8$ - $^5/_6$
5. 3 $^1/_9$ - $^2/_5$
6. 2 $^1/_6$ - 1 $^2/_3$
7. $^7/_8$ - $^1/_8$
8. 2 $^7/_9$ - $^1/_9$
9. $^4/_7$ - $^4/_7$
10. 1 $^1/_2$ - 1 $^1/_2$
11. 3 $^1/_8$ - $^3/_8$
14. $^3/_4$ - $^1/_9$
15. $^1/_4$ - $^1/_6$
16. $^5/_6$ - $^5/_9$
17. 2 $^2/_3$ - 1 $^7/_9$
18. 5 $^3/_4$ - 1 $^5/_6$
19. 2 - $^3/_4$
20. 4 - 2 $^7/_8$
21. 1 $^7/_8$ - 1
22. 5 $^1/_2$ - 3

## Appendix D

## Systematic Erroneous Procedures Observed for Addition of Fractions

1. The child adds both numerators and both denominators to obtain the numerator and the denominator of the sum respectively. This bug alone constituted 73% of all the bugs observed for the addition test; in agreement with Lankford (1972) when he notes that "this is the most prevalent practice" observed.
   Example: $^3/_4 + {}^1/_5 \rightarrow {}^4/_9$.
2. The child adds all the values of the first term (which may be fractional, whole or both) and records this value. The same process is done for the second term. Then, the smaller value is put over the larger to provide the sum.
   Example: $1\,^3/_4 + {}^4/_7 \rightarrow {}^8/_{11}$.
3. When either or both of the terms are mixed, the whole numbers are added to the sum of the numerators and this value is recorded. The denominators are also summed. The sum of the fraction is taken to be the former value over the latter.
   Example: $1\,^2/_4 + 1\,^1/_2 \rightarrow {}^{(1+2+1+1)}/_{(4+2)} \rightarrow {}^5/_6$.
4. The child does not cancel in the final answer. Example: $^4/_8 + {}^2/_8 \rightarrow {}^6/_8$ (should cancel $\rightarrow {}^3/_4$).
5. The child multiplies instead of adding.
   Example: $^1/_4 + {}^1/_6 \rightarrow {}^1/_{24}$.
6. The child becomes so engrossed in the adding the fractional parts that he/she forgets to add the whole numbers.
   Example: $9\,^1/_3 + 5\,^5/_9 \rightarrow {}^3/_9 + {}^5/_9 \rightarrow {}^8/_9$.
7. The child has learned a mechanical procedure for changing two fractions so that they have the same denominator. He/she adds the *unlike* denominators to get the common denominator, then he/she will multiply the numerator and denominator of the first fraction to get the numerator for the second fraction. Similarly, the numerator and denominator of the second fraction are multiplied

to get the numerator of the first fraction. He/she then proceeds to add the fractions.

Example: $^2/_5 + {}^1/_2 \rightarrow {}^2/_7 + {}^{10}/_7 \rightarrow {}^{12}/_7$.

8. The child finds the least common denominator but still maintains the original numerators without enlarging them.

   Example: $^3/_4 + {}^1/_2 \rightarrow {}^3/_4 + {}^1/_4 \rightarrow {}^4/_4 \rightarrow 1$ (note LCM = 4).

9. The child successfully finds the least common denominator and proceeds to write down the equivalent fractions but still goes on to add the numerators to provide the numerators of the sum and does likewise for the denominators.

   Example: $^1/_4 + {}^3/_8 \rightarrow {}^2/_8 + {}^3/_8 \rightarrow {}^5/_{16}$.

10. When either of the terms is a loose whole number, it is added to the numerator to provide the numerator of the answer and likewise for the denominator.

    Example: $2 + {}^2/_3 \rightarrow {}^{(2+2)}/_{(3+2)} \rightarrow {}^4/_5$.

11. The child adds all the values of the first term (which may be fractional, whole or both) and records this value. The same process is done for the second term. Then, the first recorded value is placed over the second to provide the sum.

    Example: $^3/_5 + {}^2/_3 \rightarrow {}^{(3+5)}/_{(2+3)} \rightarrow {}^8/_5$.

12. When either or both of the terms are mixed, the whole numbers are added to the sum of the denominators and this value is recorded. The numerators are also summed. The sum of the fraction is taken to be the latter value over the former.

    Example: $2\,{}^1/_4 + 1\,{}^1/_2 \rightarrow {}^{(1+1)}/_{(2+1+4+2)} \rightarrow {}^2/_9$.

13. The numerators and denominators are added as described in procedure (1) but the pupil eventually gets the quotient of the latter sum over the former sum and adds this to the loose numbers (if available).

    Example: $^3/_8 + 3\,{}^1/_8 \rightarrow {}^4/_{16}$ (after summing numerators and denominators).

    $^{16}/_4 + 3 \rightarrow 7$ (note that 3 is the original loose number).

14. The child gives up with the problem where the first term is a loose number.

    Example: $4 + {}^1/_4 \rightarrow$ <no answer>.

15. The child gives up with the problem where the second term is a loose number.

    Example: $^1/_4 + 2 \rightarrow$ <no answer>.

16. The child calculates the correct LCM but enlarges the numerators by cross multiplying. It is interesting to note that this results in the correct answer when both denominators are different and prime.

    Example: $^1/_4 + {}^5/_6 \rightarrow {}^6/_{12} + {}^{20}/_{12} \rightarrow {}^{26}/_{12}$.

17. The child does not unvulgarise in the final answer.

    Example: $^3/_4 + {}^2/_4 \rightarrow {}^5/_4$ (should unvulgarise to produce 1 $^1/_4$).

18. If neither term is mixed then the denominators are always multiplied to provide the denominator of the sum. The numerators on the other hand are always added to provide the numerator of the sum.

    Example: $^5/_7 + {}^8/_9 \rightarrow {}^{13}/_{72}$.

19. A common denominator (sometimes the least) is calculated but the process of finding the equivalent fractions is erroneous. The child correctly divides the denominator of each fraction into the common denominator but adds, instead of multiplying, this quotient to the numerator of that fraction.

    Example: $^1/_6 + {}^1/_3 \rightarrow {}^2/_6 + {}^3/_6 \rightarrow {}^5/_6$.

## Appendix E

## Systematic Erroneous Procedures Observed for Subtraction of Fractions

1. The child first finds the difference between the whole numbers, if they exist (if there is no whole number in a term, the value of the whole part is taken as zero). Then the child proceeds to record the difference of the numerators and the difference of the denominators as the numerator and the denominator of the difference respectively. (Note that the child totally ignores the order of the subtrahend and the minuend when doing the subtraction process.
Examples: $6\,{}^{2}/_{3} - 3\,{}^{1}/_{6} \rightarrow 3\,{}^{1}/_{3}$; $4\,{}^{5}/_{8} - 1\,{}^{3}/_{4} \rightarrow 3\,{}^{2}/_{4}$.
2. The child finds the LCM but does not enlarge the numerators but just goes to find their difference.
Example: ${}^{5}/_{6} - {}^{1}/_{3} \rightarrow {}^{5}/_{6} - {}^{1}/_{6} \rightarrow {}^{4}/_{6} \rightarrow {}^{2}/_{3}$.
3. The child does not cancel in the final answer. Example: ${}^{5}/_{8} - {}^{1}/_{8} \rightarrow {}^{4}/_{8}$ (should reduce $\rightarrow {}^{1}/_{2}$).
4. The child does as described in (1) but does not ignore the order of the minuend and the subtrahend.
Example: $6\,{}^{1}/_{3} - 3\,{}^{1}/_{6} \rightarrow 3\,{}^{1}/_{-3}$.
5. When borrowing, the child adds ten to the numerator as though he/she were doing integer multi-column subtraction. Note that this can still result in the correct answer if the denominator is ten.
Example: $3\,{}^{1}/_{9} - {}^{2}/_{5} \rightarrow 2\,{}^{11}/_{9} - {}^{2}/_{5} \rightarrow 2\,{}^{9}/_{4}$.
6. Where only one fraction appears in the problem, the whole numbers are subtracted as normal in the mixed algorithm. The fraction though is just copied to the answer irrespective of whether it exists in the minuend or in the subtrahend.
Example: $4 - 2\,{}^{7}/_{8} \rightarrow 2\,{}^{7}/_{8}$.

7. When borrowing the pupil subtracts one from the loose number without properly adding an equivalent amount to the fraction. In every case the numerator of the fraction is crossed out and the same number as the common denominator is written as the new numerator (rather than being added to the original numerator ).
   Example: $4\ ^2/_8 - 1\ ^1/_2 \rightarrow 3\ ^8/_8 - 1\ ^4/_8 \rightarrow 2\ ^4/_8$.
8. The child tries to subtract as described in procedure (1) but if the numerator/denominator of the first fraction is smaller than the respective numerator/denominator of the second fraction then he/she records a zero as the numerator/denominator of the difference.
   Example: $1\ ^3/_4 - ^5/_6 \rightarrow 1\ ^0/_0$.
9. The child systematically omits to write down the whole number in the difference.
   Example: $3\ ^1/_8 - ^3/_8 \rightarrow ^6/_8$.
10. The pupil carries out the procedure described in procedure (1) but writes nothing down when the numerator/denominator of the first fraction is smaller than the respective numerator/denominator of the second fraction.
    Example: $1\ ^3/_4 - ^2/_5 \rightarrow 1\ ^1/$<space>.
11. The child does not decrement the whole number after borrowing.
    Example: $4 - 2\ ^7/_8 \rightarrow 2\ ^1/_8$.
12. The child does not borrow. He/she calculates a common denominator (not necessarily the least), enlarges the numerators correctly but subtracts ignoring the fact that one of them is the minuend and the other is the subtrahend.
    Example: $2\ ^1/_4 - 1\ ^3/_8 \rightarrow 1\ ^{(2-3)}/_8 \rightarrow 1\ ^1/_8$.
13. The child adds instead of subtracting.
    Example: $^1/_2 - ^1/_4 \rightarrow ^3/_4$.
14. The child subtracts the fractional parts as described in procedure (1) but the loose or whole parts are added up instead.

Example: $5\ ^{3}/_{4} - 1\ ^{5}/_{7} \rightarrow 6\ ^{2}/_{3}$.

15. The child does not change $^{0}/_{N}$ to 0 in the final answer.

    Example: $^{1}/_{4} - ^{1}/_{4} \rightarrow ^{0}/_{4}$.

16. The child writes things like $0\ ^{N}/_{D}$ instead of $^{N}/_{D}$ where N and D are integers.

17. Where ever the answer becomes $^{0}/_{N}$, where N is an integer value, the child changes it to $^{1}/_{N}$.

    Example: $^{1}/_{2} - ^{1}/_{2} \rightarrow ^{0}/_{2} \rightarrow ^{1}/_{2}$.

18. The child changes $^{0}/_{D}$ to $^{N}/_{D}$ where D is the denominator in the difference and N is equal to the numerators.

    Example: $^{5}/_{7} - ^{5}/_{7} \rightarrow ^{5}/_{7} \rightarrow ^{5}/_{7}$.

19. Where ever the answer is of the form $^{0}/_{D}$, the child writes down D as the answer.

    Example: $^{1}/_{8} - ^{1}/_{8} \rightarrow ^{0}/_{8} \rightarrow 8$.

20. The child does unvulgarise in the final answer.

    Example: final answer = $^{4}/_{3}$ (should unvulgarise to produce $1\ ^{1}/_{3}$).

21. The least common denominator is calculated but the child cross multiplies to enlarge the fractions.

    Example: $^{2}/_{3} - ^{1}/_{6} \rightarrow ^{((2\text{x}6) - (1\text{x}3))}/_{6} \rightarrow ^{9}/_{6}$.

# Appendix F

# User Manual

This appendix is intended to provide some brief information on how to go about using or running FITS. It is fairly straightforward but in case of any difficulty (e.g. on how to load various software), appropriate manuals should be consulted or help could be sought from others knowledgeable with Quintus Prolog/Suntools on SUN workstations.

FITS runs under Quintus Prolog on a SUN-3 workstation; SUN workstations run under UNIX. Suntools is SUN's windowing environment which jointly with Quintus Prolog, has made SUN workstations a favourite amongst AI researchers. Suntools has to be initially loaded once logged on to the SUN by typing in the following command:

```
$    suntools
```

Once in Suntools, the ensuing process is next followed:

- The student should provide answers on paper to the pre-test questions stored in a file *sums_0*. These answers are entered and stored in a new file (note that the file name should be unique, preferably bearing some relation with the name of the student) such that this file is similar to *sums_3* (which is an example student answers file). This file of answers for the particular student is used for pre-modelling.
- The module *sm_init* is next loaded and a new clause must be be added to the *filename/1* clauses to introduce the particular student's file of answers to FITS. For example, if the student's file of answers is *janes_file* then the Prolog clause

  ```
  filename(janes_file).
  ```

  is inserted into this module at the same position as the other *filename/1* clauses.

This whole process is done only once per student, i.e. the first time the student logs on to FITS. The topmost level module, *top_level*, is next loaded and interpreted. This takes about 3 minutes.

Once this top level module has been loaded and interpreted, and the student's pre-modelling file set up, FITS is now ready to be activated. It is activated by typing in the following command:

```
?-  start_FITS.
```

The student now proceeds to using the tutor and is expected to type in the name of his/her file of answers when asked early in his/her interaction with the tutor. The student can quit anytime he/she wishes but is expected to use the same *registration name* that was used when he/she initially logged on to FITS at every other subsequent 'login' time. FITS can be reactivated (if still in Suntools) by using the same command as above; it will consult the student's records and proceed from where he/she quitted.

However, if the student has exited Suntools and wants to log on to FITS again (i.e. he/she is using FITS for the second time or more), then Suntools and the top_level module will have to be reloaded (top_level module will have to be re-interpreted) and FITS activated as before (remember the process described above for setting up a pre-modelling file for the student is NOT repeated). FITS will proceed with tutoring from where the student last quitted.

# Appendix G

## System Modules

This appendix provides some brief information on the modules that comprise FITS to help anyone who might wish to examine it in more detail.

- **top_level** is the topmost level module from which the entire ITS is activated. It is FITS's main user interface module.
- **sm_init** is the module that creates and initialises the student model/history files through pre-modelling.
- **pre_teach** is the module that (re)teaches pre-requisite skills.
- **quiz_pr** is the module that quizzes the students on the taught pre-requisite skills.
- **malrules** is the bug catalogue module.
- **utilities** is a module containing various predicates globally used in other modules.
- **fraction_kb** is the fractions knowledge base module.
- **ts_strategy** is the global tutoring strategy module.
- **generator** is the module which generates various problems of differing difficulty levels.
- **expert_system** is FITS's expert fractions problem solver module. It contains the bon-rules.
- **ps_monitor** is FITS's problem solving monitor module which does the bulk of the model-tracing and student modelling.
- **analyse_shfile** is a module that analyses a student-tutor history file in order to update the student model.
- **sums_0** contains the pre-test questions.
- **sums_3** is an example test student answers file used for pre-modelling.
- **users** contains the all FITS's registered users.

- ***.sh** (where * instantiates to any student's name) is *'s student-tutor history file. For example *john.sh* is John's student-tutor history file.
- ***.sm** is *'s student model.
- **ideal_sm** is FITS's ideal student model.

## Appendix H

## Student Comments

These are some of the comments the students made about FITS. These are either copied mostly *verbatim* from their written comments or constructed from the notes made from their commentaries while they were using the system. Every effort was made to keep the constructed comments as close as possible to what was originally said.

- Generally, I thought the program itself was good. It has made me feel more confident than I was before.
- It was good the way the tutor praised me when I did something right.
- I also liked the way it clearly explained the workings of the sum that it was doing.
- I think the program would be good for beginners who want to learn to add fractions but it is also good for people like myself who want to improve.
- I enjoyed doing it.
- I did not like the full stops.
- I did not like how when I had reached near the end of the page, it suddenly jumped to the top.
- Some of the things printed were difficult to understand.
- I liked the way it helped me when I was in trouble.
- It is good the way it went back to teach something I have forgotten.
- It is good how it can do very difficult sums which even my teacher cannot do.
- As you use it it becomes easier.
- The tutor has made me see that, to do a difficult sum, you just do a few operations which make it simple so that I can do it easily.
- I enjoyed the way it really sometimes made things easy for me.

- I like the way it correctly tells me what I have done wrong.
- I also like the way it shows me some examples of what to do when I am not sure.
- I hated those full stops and that jumping screen. Sometimes I got lost because of it.
- I do not like the way it was teaching me how to cancel fractions when I already know how to do it.
- Well, it was a bit long.
- I like the comments.
- I like the way it works out the sums showing me where and when to use the various skills.
- It gets harder so I found it difficult.
- The sums is for ages between 9-12.
- My elder brother who is 14 can use it as well. He is poor in fractions.
- The sums which were hard before were easy when I finished from the tutor. I now understand fractions better.
- Maybe you could put some nice pictures to make it more attractive.
- That screen was awful sometimes.
- That part where it does many sums showing a new thing each time is good. It makes me understand fractions better.
- I can now do some of the sums I got wrong before I started.
- I have used a computer program like this before, but this is a lot better. It can even work the sums out.
- I really enjoyed it. Can you improve it so that I can come back and use it?
- Student: How long did it take you to write this program?

  Author: About 1 year but I had been thinking about it for 1 $^1/2$ years. So altogether 2 $^1/2$ years.

  Student: Just this!!