AN EXPERT SYSTEM FOR EDUCATIONAL DIAGNOSIS USING THEORIST

by

JAMES BRUCE TUBMAN

CS-86-32

A thesis

presented to the University of Waterloo

in fulfilment of the

thesis requirement for the degree of

Master of Mathematics

in

Computer Science

Supervised by Marlene Jones

Waterloo, Ontario. 1986

Abstract

An Expert System for Educational Diagnosis Using THEORIST

During the past decade, many diagnostic expert systems have been developed; the majority of these are rule-based production systems. In this thesis, we explore an alternative approach. We present the fundamentals of THEORIST (Poole et. al 1985), a logic programming system that uses a uniform deductive reasoning mechanism to construct explanations of *observations* in terms of *facts* (or rules) and *hypotheses* (which are defaults). Then we describe a small expert system within the domain of educational diagnosis which has been implemented using THEORIST. Future research plans are discussed in light of the results of this initial experimentation with THEORIST.

CONTENTS

LIST OF FIGURES

CHAPTER 1:

EXPERT SYSTEMS

1. INTRODUCTION

The introduction of computers to schools has led to their application in many areas. Most commonly, they are used in Computer Assisted Instruction (CAI), Computer Literacy, Computer Managed Instruction, and Mathematics. The work of Jones and McLeod (1983) indicated that computer applications in the area of Special Education (that is, the education of children with learning problems) held great potential for making educational diagnostic expertise much more readily available to schools, particularly in rural and remote areas. Jones (Colbourn 1982) built a prototype expert system for the diagnosis of reading difficulties. This system was a traditional expert system, written in Lisp using production rules.

This research represents an extension of Jones' work, using a new approach. The approach is the use of a logic programming system called THEORIST. The goal of this research is to determine whether the THEORIST paradigm is an appropriate foundation for the development of expert systems for educational diagnosis. To determine this, a small expert system for the diagnosis of arithmetic problems was designed, built, and tested.

2. DEFINITION OF AN EXPERT SYSTEM

An expert system is a computer-based system that helps solve difficult problems in specific fields, such as medicine, science or engineering (Barr and Feigenbaum 1982; Hayes-Roth, Waterman and Lenat 1983; Winston and Prendergast 1984; Buchanan and Shortliffe 1984). The field that the program deals with is called its *domain*. The problems that expert systems solve are ones that highly trained human experts solve using large amounts of *heuristic*, or "rule of thumb", knowledge, in addition to algorithmic methods. Expert systems are usually written in a style that separates this heuristic knowledge from the program's control structure. The program reasons with this knowledge to reach its conclusions. In the course of its reasoning, the system may interact with the user: the system may ask the user to supply information necessary to reach the conclusion, and the user may ask the system to explain its reasoning. An example of an expert system is a program that gives consultative advice on the medical diagnosis of infections, using a separate body of heuristic knowledge gleaned from physicians and questioning the user to obtain necessary information about the case at hand. A program that typesets text is not an expert system, because, although the humans who set type are experts, the methods that such a program employs are algorithmic, not heuristic, and do not require interaction with the user.

3. HOW A TRADITIONAL EXPERT SYSTEM WORKS

A typical expert system (Hayes-Roth, Waterman and Lenat 1983; Buchanan and Shortliffe 1984) represents its knowledge in the form of *production rules*. A production rule is a small executable program segment that consists of a set of *conditions* and a set of *actions*. For a rule to be considered applicable in a given situation, all of its conditions must be satisfied. If the rule is applicable, then its set of actions may be executed. Thus a production rule is similar to an IF-THEN statement in a typical programming language. To represent a rule meaning, "If Eric has embezzled money, and the police suspect Eric, then he should fly to Rio," one might use a production rule of the form:

```
(IF     (AND   (EMBEZZLED ERIC MONEY)
               (SUSPECT POLICE ERIC))
 THEN   (FLY-TO ERIC RIO))
```

The number of production rules used by an expert system can range from a few dozen to the hundreds. These rules are usually gathered together in a structure called the *knowledge base*. This is where the domain-specific knowledge of the system is concentrated, separating it from the inference and control mechanism. In addition to production rules, the knowledge base may contain facts about the entities in the domain, and descriptions of relationships between these entities.

This knowledge is used by the program's inference and control structure. This structure determines which rules are appropriate in a given situation, resolves conflicts among rules, and executes the action portion of the appropriate rule. Conflict arises when the conditions of several different rules are all satisfied.

There are several ways to handle such a conflict. The easiest way is to choose the first rule in the knowledge base in which all conditions are satisfied. A better way is to associate a heuristic value with each rule, and choose the applicable rule with the best heuristic value. The heuristic value may be an estimate of the cost or utility of carrying out the actions in the rule. In some cases, the number of applicable rules may be so large that it is beneficial to use a *planner* (Sacerdoti 1977) to evaluate what rule to use next. The DENDRAL expert system for chemical analysis (Lederberg 1964; Lindsay et al. 1980) uses planning to reduce the number of possible molecules it must consider while trying to determine a molecule's structure. The method chosen for handling a conflict depends upon the domain of the expert system.

In response to each query from the user, the expert system continuously evaluates the conditions in the rules in its knowledge base and takes the actions indicated by these rules. As the evaluation proceeds, the system may ask the user for input that can help its reasoning. The process stops when the desired conclusion is reached, or no conclusion can be reached. The user may then query the system again, ask it for an explanation of its conclusion, or add new information to the knowledge base.

An expert system may have other features to increase its usefulness and to give confidence to its users. One of the most important of such features is a *justification facility*. If the user wants to know how a conclusion was reached, or why the system wants to know a particular fact, he or she asks the expert system to explain. When the user asks "why", the system will explain the decision it is trying to make, for which the information it has requested is relevant. In a diagnostic expert system, the diagnostician may wish to know why a certain piece of information is wanted before an expensive or time-consuming test is administered. To answer the "how" question, the system traces its chain of reasoning, showing each decision that was made. This gives the user confidence in the decision that the system has reached.

Another useful feature for an expert system is a natural language front end. This allows the user to interact with the system using a restricted dialect of English (or other human language). The interface parses and interprets the user's commands, questions, and other input. Such interfaces can be difficult to build, especially for complex domains. Natural language interfaces are discussed in (Schank and Colby 1983; Charniak and Wilks 1976; Waltz 1978); a natural language interface for an educational diagnostic expert system is discussed in (Kindersley 1986).

In addition to querying the user for data regarding a case at hand, some expert systems allow users to add their own domain knowledge to the knowledge base. For example, some commercially available expert system shells contain an induction mechanism for extracting rules from examples. For some domains, this is a useful technique. However, the user must be careful to choose appropriate examples, or the induction mechanism may make faulty generalizations. Induction systems are not a common component of most expert systems, but they have begun to appear in some expert system shells, such as *Expert-Ease*. Expert systems based on induction are explained in (Human Edge Software 1984); the pitfalls of induction systems are discussed in (Hayes-Roth, Waterman and Lenat 1983).

4. EARLIER RESEARCH

Research in the area of expert systems has been going on since the 1960's. Most early expert systems were custom built for each application by programmers called *knowledge engineers*; this is still true for many systems. Recently, software products called *expert system shells* (such as *OPS-5, M1, S1,* and *Expert-Ease*) have been developed. Such shells provide knowledge engineers with better tools for building expert systems; in some cases, they allow end users to build their own expert systems. Expert system shells contain a control structure that is usually modelled on the control structure of the earlier custom built expert systems; the user adds his or her own domain-specific knowledge.

Many noteworthy expert systems have been implemented; some of them embody ideas that influenced the expert system described in this thesis. MYCIN (Shortliffe 1976) is a consultative expert system for diagnosing and recommending therapy for infectious diseases. It is notable for its separation of its knowledge base from its control structure, and for its ability to reason under conditions of uncertainty using numerical quantities called *certainty factors*. Associated with MYCIN is the comprehensive knowledge acquisition, debugging, and explanatory system called TEIRESIAS (Davis 1976). It lets a user find out how and why MYCIN has reached a conclusion. If the conclusion is erroneous, it helps the user modify the knowledge base to ensure correctness. Another medical diagnosis expert system is CADUCEUS, formerly known as INTERNIST (Pople 1977). It is a consultation program for the domain of internal medicine. Given a list of manifestations of a disease, it formulates diagnoses by forming hypotheses about the disease. Knowledge is represented by a *disease tree*, which puts more general diseases near the root and more specific diseases at or near the leaves. The knowledge representation scheme of CADUCEUS strongly influenced the design of the expert system described in this thesis.

5. INADEQUACIES OF PREVIOUS EXPERT SYSTEM DESIGNS

Previous generations of expert systems, and the expert system shells that followed them, have proven useful in many areas, but they still suffer from some weaknesses. The most serious of these is how they deal with uncertain knowledge. Usually, uncertainty is dealt with via some kind of numerical measurement of uncertainty, such as MYCIN's certainty factors. However, developers of such systems are usually vague about how the measurements are determined. The precise determination of these numbers is usually made during a "tuning" phase. The knowledge engineer keeps changing the numbers until the right answers emerge. The "magic numbers" are determined by trial and error, rather than by a method that could properly be called scientific.

An alternative to reasoning with numerical uncertainty is *default reasoning*. At the University of Waterloo, researchers have developed a default reasoning system called THEORIST (Jones and Poole 1985; Poole et. al 1986), that is modelled on scientific reasoning. The expert system described in this thesis was based upon this model.

6. THESIS OVERVIEW

This thesis describes the design and implementation of an expert system whose domain is the educational diagnosis of arithmetic difficulties in elementary school children. The system uses THEORIST as its inference mechanism. The implementation is intended to investigate the hypothesis that the THEORIST paradigm is a suitable foundation for expert systems development.

Chapter 2 describes the THEORIST inference mechanism, including its mathematical background, details of operation, knowledge base structure, and current implementation state. The domain of educational diagnosis is discussed in Chapter 3; educational diagnosis is defined, diagnostic procedure is explained, and previous computer applications to educational diagnosis are discussed. Chapter 4 discusses the implementation and design decisions for the expert system, including the structures and purposes of the main software components, and the restrictions made on the domain. The flow of control, contents of the knowledge base, and sample interactions with the system are examined in Chapter 5. In Chapter 6, testing of the system on both clinical and non-clinical test cases is discussed, along with the implications of test results. Finally, Chapter 7 discusses considerations for future versions of educational diagnostic expert systems, and presents the conclusions of the thesis.

CHAPTER 2:

THEORIST

7. *MATHEMATICAL BACKGROUND*

Researchers at the University of Waterloo have proposed that expert system development be based on the philosophy of constructing and reasoning with scientific theories (Poole 1984; Jones and Poole 1985; Poole et. al 1986). This philosophy has been implemented as the THEORIST system.

THEORIST is a logic programming system which attempts to explain observations in terms of theories developed from *facts* and *defaults* (also called *hypotheses*). THEORIST distinguishes facts from hypotheses and attempts to use deduction to construct consistent theories (made of instances of hypotheses) for which the observations are logical consequences. A theory developed by THEORIST should predict the given observations and be empirically testable, by verifying whether other predictions based upon this particular theory are valid for the test case at hand. For diagnostic systems, the diagnosis is the theory of what the problems are; thus THEORIST is particularly appropriate for such systems.

7.1 Default Logic

A THEORIST knowledge base consists of three types of logical formulae: *facts* (F), possible *hypotheses* (Δ), and *observations* (G). To explain the observations, the THEORIST reasoning strategy attempts to accumulate sets of facts and instances of hypotheses. These sets must be consistent, and must logically imply the observations. The formulae used by THEORIST are:

F **Facts** — The set of formulae which are known to be true in the world that the system is trying to represent. Facts are assumed to be consistent. There are two kinds of facts: general knowledge about the domain, which is true in all cases (often referred to as rules); and knowledge regarding the particular case at hand (sometimes called case facts). Not all facts need to be specified initially; some of them may be acquired during the course of the diagnosis by asking the user and asserting the user's response.

Δ **Defaults or Hypotheses** — These are the possible hypotheses which may be used to explain the observations. Defaults may have free variables, in which case instances of the defaults can be made by binding values to any or all of the free variables.

G **Observations** — The set of goals to be explained by the theories developed by THEORIST.

To illustrate what these facts and defaults look like, here is a simple THEORIST knowledge base from the domain of bird-watching. (*n* is how THEORIST represents the logical *not*.)

```
fact     bird(X) <- penguin(X);
fact     bird(X) <- woodpecker(X);

fact     penguin(chilly-willy);
fact     woodpecker(woody);

fact     n(fly(X)) <- penguin(X);

default  fly(X) <- bird(X);
```

This represents the following facts: penguins and woodpeckers are birds; Chilly Willy is a penguin; Woody is a woodpecker; and penguins can't fly. The default represents the hypothesis that something can fly if it is a bird.

A theory is a consistent subset of possible hypotheses that imply the observations. More formally, G is *explainable* if there is some subset D of Δ such that

$$\begin{array}{ll} F \cup D & \models G \\ F \cup D & \text{is consistent.} \end{array}$$

The consistent subset of hypotheses D should be perceived as a "scientific theory" (Popper 1959; Quine and Ullian 1978) that is used to explain G. In the bird-watching example, THEORIST can explain the observation *fly(woody)* with the theory *fly(woody) <- bird(woody)*; this is consistent with the facts that Woody is a woodpecker and that woodpeckers are birds. However, based on this knowledge base, THEORIST cannot explain the observation *fly(chilly-willy)*, because this is inconsistent with the fact that penguins cannot fly.

7.2 Developing Hypotheses

The THEORIST theorem prover is goal-directed, and can answer questions about whether something follows from a set of facts and hypotheses. Unlike resolution systems, it preserves the input structure supplied by the user, and can thus explain its reasoning in the terms employed by the user. Unlike a Prolog system, it is not restricted to using Horn clauses; it can use arbitrary conjunction, disjunction, equivalence and negation.

Theories are developed by using facts F and defaults Δ as axioms in a proof of the observations G. The system attempts to explain the observation by using what is known (the facts), and by postulating hypotheses (the defaults). A potential theory D is composed of instances of the defaults used in the proof. Once the proof has succeeded, D is known to be adequate to explain G. THEORIST must then show that $F \cup D$ is consistent. $F \cup D$ can be proven inconsistent if it can be shown that $F \cup D \models \neg d$ for some $d \in D$. The theory is not consistent if such a proof succeeds. If all such proofs fail, the theory is consistent. The attempt to prove inconsistency is analogous to empirically verifying a scientific theory by allowing it to make predictions, then comparing these predictions to what is known.

This process can be illustrated with the bird-watching example. To explain the observation that Woody can fly, the user asks THEORIST "*explain fly(woody)*". The system consults the knowledge base and finds that there exists a default that can explain whether Woody can fly, namely "*fly(X) <- bird(X)*". It binds the name "woody" to the variable *X* then tries to prove that Woody is a bird. This is done with the facts "*bird(X) <- woodpecker(X)*" and "*woodpecker(woody)*". Then THEORIST checks for consistency: it tries to see if the union of the facts and defaults imply the negation of some default used in the proof of *fly(woody)* (in this case there was only one, *fly(X) <- bird(X)*). This proof fails, so $F \cup D$, which consists of *fly(woody) <- bird(woody), bird(woody) <- woodpecker(woody)*, and *woodpecker(woody)*, is consistent.

The observation that Chilly Willy can fly cannot be explained from this knowledge base. THEORIST attempts to prove "*explain fly(chilly-willy)*" in the same way it did for Woody: with the default "*fly(X) <- bird(X)*" and the facts "*bird(X) <- penguin(X)*" and "*penguin(chilly-willy)*". The process fails during the consistency check; the fact "*n(fly(X)) <- penguin(X)*" implies the negation of the default "*fly(X) <- bird(X)*", which was used in the construction of the proof.

For the domain of educational diagnosis, the defaults are possible hypotheses that are acceptable in a diagnosis. In the system described in this thesis, such hypotheses are about whether a skill has been mastered or not, and by what means this can be determined. The facts are what is known about the case at hand: the child's name, age, grade level, whether the child has taken a given test, and so forth. If the fact is not known, the system can be authorized to ask the user for this information; for this domain, most facts are obtained in this fashion.

The system builds a theory about what the child knows or does not know, and what problems he or she is exhibiting. The theory can be validated by comparing it to the child's actual performance. The ability of the system to ask the user for information allows the system to automatically propose experiments to test the child, so the diagnosis can be refined as it discriminates between possible hypotheses. The system carries out this process by querying the user for the results of standardized diagnositic tests, which indicate the child's level of performance.

8. *HYPOTHESIS ABSTRACTION HIERARCHY*

Expert systems in complicated domains will have many possible hypotheses. Large numbers of hypotheses can lead to combinatorial explosions as THEORIST attempts to select appropriate hypotheses in the process of building its theories. In order to reduce the problem of combinatorial explosion, hypotheses can be arranged into an *and/or* tree called the *hypothesis abstraction hierarchy* (Jones and Poole 1985). An example of such a tree, based upon the domain of diagnosing elementary arithmetic difficulties, is shown in Figure 1.

The *and-nodes* correspond to concepts that can be broken down into components. For example, in Figure 1, node *task* is an and-node, indicated by the arc through the branches descending from it. A theory is better if it contains more components of an and-node. *Or-nodes* have descendents that are disjoint alternatives that form more specific cases. This is also illustrated in Figure 1; the node labeled *process* is an or-node, with disjoint descendents *addition*, *subtraction*, *multiplication*, and *division*. The better theory is one with fewer of these alternatives. A node may have various facts associated with it; such facts are information that is relevant to the particular node. An extensive example is given in Chapter 5.

To perform a diagnosis, the system moves down the tree, from the root towards the leaves, forming theories that are more and more specific; that is, the theories explain the observations at greater and greater levels of detail. Inappropriate subtrees, which could not possibly be part of the diagnosis, are not considered. If there are no apparent difficulties with a particular skill, its subskills will not be investigated. Hypotheses are only considered if they are potentially relevant. For example, in the domain of arithmetic diagnosis, if the child is not experiencing any difficulties with processes in general, then the specializations

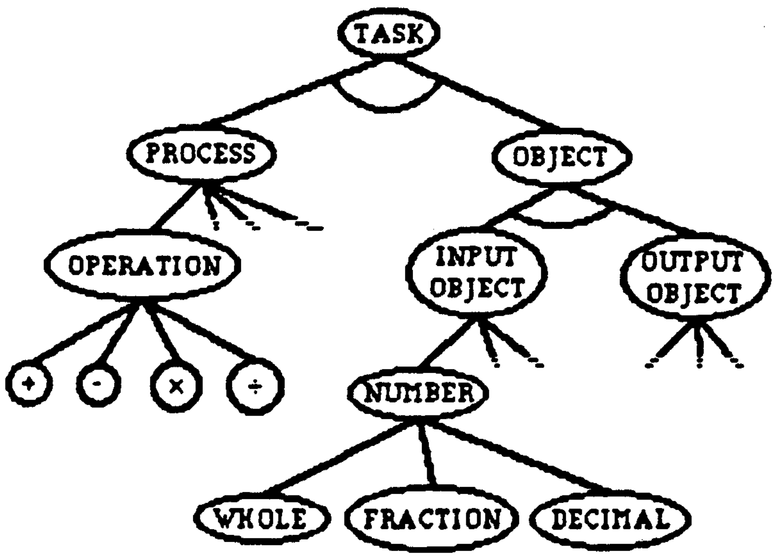


Figure 1. Hypothesis Abstraction Hierarchy for Elementary Arithmetic.

of process (addition, subtraction, multiplication, and division) will never be considered.

On the other hand, if the child is experiencing difficulty with a particular skill, the system will encounter the node corresponding to that skill in the hierarchy. If the node is not a leaf node, it will have descendents, one or more of which may be considered as candidate hypotheses. The results from the processing of the current node determine which, if any, of these descendents will be processed.

The process of descending the tree may terminate for several reasons: the necessary information may not be available; the skill may not be appropriate for the age or grade level of the child; or the theories formed may indicate that further assessment will not yield additional information. Further assessment of the specializations of a task is terminated as the result of executing special rules called *pruning rules*. Each task has its own set of pruning rules associated with it. The pruning rules for a given task determine, based on the theories formed for that task, whether the search of the hierarchy should be pruned, and at what point. Therefore, the hierarchy provides a means of modeling the problems of the child at a level appropriate to the circumstances of the individual diagnosis.

9. CURRENT IMPLEMENTATIONS

A preliminary version of THEORIST, denoted Simple THEORIST, has been implemented in Waterloo UNIX Prolog (WUP) (Poole et. al 1986). Simple THEORIST compiles THEORIST statements into Prolog, which is then executed like a regular WUP program. This initial implementation is not intended as a final product but rather as a tool with which to experiment.

Although Simple THEORIST can be used to develop theories, it does not have the power of true THEORIST. It does not have a law of the excluded middle, nor does it automatically produce the contrapositive of its rules; if the contrapositives are to be used, they must be entered explicitly. Also, it does not have dependency-directed backtracking; normal Prolog backtracking is used instead. Simple THEORIST does not have a theory comparitor, so it is unable to select the "best" theory; it can quit after the first theory, or it can be prompted by the user to look for another theory. The version of Simple THEORIST used in this expert system was slightly modified so that if an observation could not be proved, the atom *explain-fails* would be returned in place of a theory; this made dealing with failure easier. Simple THEORIST has the advantage that it can be implemented as efficiently as Horn clauses, with the consequent gain in speed.

The details of the hypothesis abstraction hierarchy also differ in Simple THEORIST. Simple THEORIST does not do the traversal of the tree by itself; a Prolog driver program does the tree traversal, invoking Simple THEORIST as necessary. The Prolog program uses the results obtained by Simple THEORIST to do the pruning of the search. In addition, the facts and defaults are not scoped in this version, but are added to the global database as they are encountered in the traversal. In practice, it was found that this did not cause any combinatorial explosion problems, since the number of appropriate theories was always small.

Despite the current limitations of Simple THEORIST, testing of the THEORIST paradigm is underway, including the development of the expert system for educational diagnosis discussed herein.

CHAPTER 3:

THE DOMAIN OF EDUCATIONAL DIAGNOSIS

10. DEFINITION

When a child in school is having unusual difficulty in learning, remembering, or using material that is taught, the child's teachers may feel it is necessary to undertake an educational diagnosis. Educational diagnosis is the detection of a learning difficulty, the determination of its nature, and its remediation (Wallace and Larsen 1979; Hammill and Bartel 1982). It can be a complex task that should be performed by qualified personnel using accurate diagnostic materials; in some circumstances, the task can be made easier if the diagnostician is guided and assisted by a computer.

Educational diagnosis can be carried out by several different kinds of people, depending upon the severity of the case at hand and the diagnostician's familiarity with the problem. At the simplest level, diagnosis can be performed by the child's regular classroom teacher. Certain kinds of diagnosis are within the qualifications of this type of teacher, who may possibly have no background in special education. Indeed, the regular classroom teacher has the advantage that he or she knows the child well and is familiar with the child's work. If problems are more severe, the child may be referred to a resource room teacher. Such a teacher has a background in special education and can carry out more complex diagnoses and recommend special remediative programs. The last level of diagnostician is the educational psychologist, who has the most comprehensive training and has the deepest knowledge of remediative methods.

To form a diagnostic conclusion, the diagnostician must take many factors into account. The first of these is the child's background: things like age, grade level, mother tongue, home life, medical and psychological problems, and so on. The child's class work, which is often the first indicator of a problem, must be examined in detail. Finally, the diagnostician may administer standardized diagnostic tests, to help determine whether there is indeed a learning problem and to scrutinize the child's skills.

Most standardized tests fall into one of two categories: *norm-referenced* and *criterion-referenced*. Criterion-referenced tests include a broad selection of items designed to measure several aspects of a skill; the results are used to determine what components of the skill need remedial action. Such tests are usually used in the late phases of the diagnosis. Norm-referenced tests are not as exhaustive but they offer statistical results which allow the diagnostician to compare the child with his or her chronological peers. Norm-referenced tests are important during the earlier phases of the diagnosis.

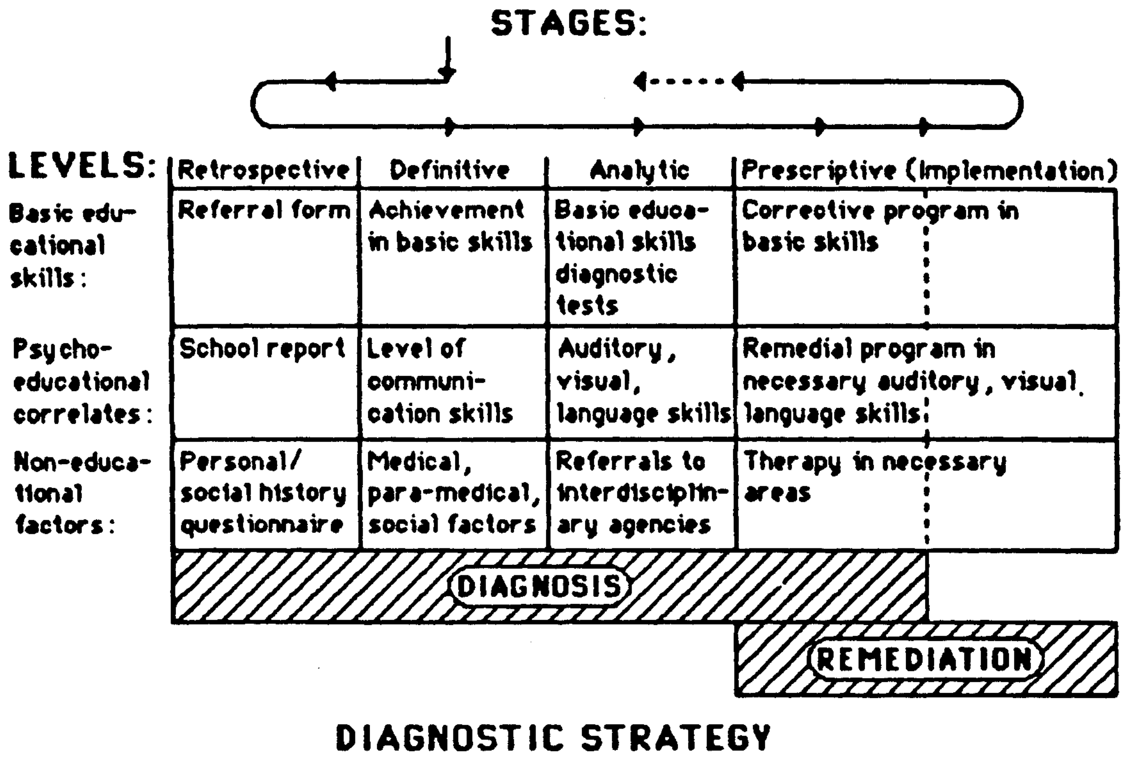
11. DIAGNOSTIC PROCEDURE

The model of diagnosis used in this system is the McLeod Educational Diagnostic Model (McLeod 1982). In this model, diagnosis is seen as a continual process, divided into four phases. (See Figure 2.)

- Retrospective: relevant data about the child's previous developmental history is reviewed. This data may be gathered from the referral form, the school report, and a personal/social history questionnaire.

- Definitive: the existence — or non-existence — of a learning difficulty is established. This is done by examining achievement in basic skill tests, the level of communications skills, and other medical, para-medical and social factors.

- Analytic: surface symptoms are subjected to progressively finer scrutiny. This is done with diagnostic tests of basic educational skills, tests of auditory, visual, and language skills, and possibly referrals to interdisciplinary agencies.



LEVELS:	Retrospective	Definitive	Analytic	Prescriptive (Implementation)
Basic educational skills:	Referral form	Achievement in basic skills	Basic educational skills diagnostic tests	Corrective program in basic skills
Psycho-educational correlates:	School report	Level of communication skills	Auditory, visual, language skills	Remedial program in necessary auditory, visual, language skills
Non-educational factors:	Personal/ social history questionnaire	Medical, para-medical, social factors	Referrals to interdisciplinary agencies	Therapy in necessary areas

Figure 2. The McLeod Educational Diagnostic Model.

- Prescriptive: corrective or remedial action is initiated. This includes a corrective program in basic educational skills, a remedial program in necessary auditory, visual, and language skills, and possibly therapy in non-educational areas.

The third and fourth stages are considered to be cyclical; the child is re-tested to measure progress in remediation, and changes to the remedial program can be made if the evidence indicates that this is necessary.

In each of these phases, several types of information should be obtained, usually by querying the user. This information can generally be classified into three categories:

- Basic educational skills, in areas such as reading, spelling, and arithmetic;
- Psycho-educational correlates, which include those intellectual, visual, auditory and language skill deficiencies that might be related to the child's problems; and
- Non-educational factors which are primarily in medical, social, and developmental areas.

The classification of data into these categories is not always well-defined, and can depend upon the diagnostician's judgement. For example, the child's emotional status may be considered at either the psycho-educational level or as a non-educational factor. What is important is that the information is incorporated; the level at which to do so is determined by the diagnostician's personal preference.

Ideally, the diagnostician begins by gathering the necessary background information about the child. Next, the diagnostician examines the child's classwork and the results of general diagnostic tests (such as tests of intellectual ability and general achievement tests) to try to confirm whether or not a problem does exist; if it does, he or she tries to find the general area of the problem. In the next step, the diagnostician administers specialized diagnostic tests to determinemore precisely the child's strenths and weaknesses in comparison to his or her chronological peers, determine his or her relative strengths and relative weaknesses in comparison to his or her other abilities, detect common error patterns, and unearth more details of psycho-educational correlates and non-educational factors. This information allows the diagnostician to plan a remedial program for the child. The child's progress is monitored, and periodic re-evaluations of the child's problems are made, followed by appropriate adjustments to the remedial programs.

12. COMPUTERS AS AIDS TO EDUCATIONAL DIAGNOSIS

Most research on applications of Artificial Intelligence to Education have been concerned with Intelligent Computer-Aided Instruction (ICAI) (Sleeman and Brown 1982; Jones 1986). This research is directed towards a different use of computers in education: using computers as aids in special education; in particular, using expert systems to assist diagnoses of children with learning problems. The potential of such systems is discussed in (Colbourn and McLeod 1983). Such a system would be able to guide a teacher (who may have relatively little diagnostic experience) through a diagnosis, drawing upon a large body of encoded diagnostic expertise to help it form a diagnosis and prescribe remediative action. For difficult cases, expert diagnosticians may find it useful to have a second opinion available from an expert system. In addition, such a system can be useful in training new diagnosticians by giving them guidance and a standard of comparison, and for keeping practicing diagnosticians up to date by making a large body of current test information available.

Expert systems have numerous capabilities that make them powerful assistants in the diagnostic process. They are able to store and retrieve large bodies of information, such as student records, data about many different diagnostic tests, algorithms and heuristics for diagnostic procedure, methods for remediation, and so on. They can process information extremely rapidly, making it possible for them to explore many different diagnostic possibilities in much less time than it would take a human to do so. Such software is also very thorough; it is not possible for an expert system to forget a step in the diagnosis procedure. The combination of these features can help a diagnostician perform a more thorough, wide-ranging diagnosis without accidently omitting or inappropriately altering a step.

Another advantage of using an expert system for diagnosis is that it can bring diagnostic expertise to schools where it is not easily available, such as rural or remote schools. Presently, if a child in such a school has learning problems, it is necessary either to bring the expert diagnostician to the child, or bring the child and a chaparone to a major centre to perform the diagnosis. In areas like northern Canada, this means that the diagnostician must be flown in or the child flown out for the diagnosis to be administered. This is time-consuming and costly. A diagnostic expert system would enable most of such diagnoses to be carried out at the child's school, using locally available microcomputers.

13. PREVIOUS COMPUTER APPLICATIONS TO EDUCATIONAL DIAGNOSIS

Although computers have been used in education for many years, their use in the specialty of educational diagnosis has been relatively recent. Important previous research in this area has been carried out by Brown and Burton, Young and O'Shea, Jones and McLeod, and McDermott.

13.1 Brown and Burton

BUGGY (Brown and Burton 1978) is a computational model of children's correct and incorrect subtraction procedures. In the BUGGY model, students' errors in subtraction are seen as "bugs": discrete modifications to an otherwise correct subtraction procedure. Based on this model, they analyzed thousands of actual children's subtraction problems with the DEBUGGY program, trying to account for each error with some combination of basic bugs. Although the BUGGY model succeeded in accounting for all the errors, the great profusion of elementary errors that were needed to account for these errors, and the flaws in Brown and Burton's basic assumptions about the nature of subtraction itself, indicate weakness in the model.

BUGGY represents arithmetic skills with a hierarchical structure called a *procedural network* (Sacerdoti 1977). An arithmetic algorithm is represented as a path through the network, hierarchically organized as a lattice. An erroneous algorithm, or bug, has a path that deviates from the path of the correct algorithm at some point. Primitive bugs are single deviations from the correct algorithm. Compound bugs are combinations of primitive bugs, up to a maximum of four. The BUGGY programs, DEBUGGY (the batch version) and IDEBUGGY (the interactive version), examine a set of subtraction problems and accumulate a set of bugs that predict all the student's errors without making any false predictions of errors. This is done by drawing upon a set of 110 simple bugs and 20 common compound bugs. Fact errors in subtraction are accounted for by a coercion scheme.

Consider an example from (Brown and Burton 1978). It shows a student's answers to five subtraction problems compared to the answers predicted by two single bugs, **0-n=n** and **smaller from larger**. A "+" denotes that the student's answer was correct; a "*" shows that the bug predicts the student's correct answer; a "***" shows that the bug predicts the student's incorrect answer.

A	*B*	*C*	*D*	*E*
45	40	139	500	312
-23	-30	-43	-65	-243
22	10	96	435	69
Student Answers				
+	+	+	565	149
0-n=n:				
*	*	*	***	***
smaller from larger				
*	*	116	***	131

In this example, the student got questions *A*, *B*, and *C* correct, and made errors on *D* and *E*. The bug **0-n=n** predicted the correct and the incorrect answers. The bug **smaller from larger** made correct predictions for questions *A*, *B*, and *D*, but incorrectly predicted answers of 116 for *C* and 131 for *E*. Since **0-n=n** consistently predicted both correct and incorrect answers, it was selected as the bug that properly explained this student's subtraction errors. In actual practice, the set of 110 simple bugs and 20 common compound bugs would all be considered as candidates for explaining the student's errors.

There are two main criticisms of the BUGGY model. The first is that there are an enormous number of primitive bugs, many of which deal with extremely specific cases. The apparent need for this great number of bugs suggests that the BUGGY model is a weak means of representing arithmetic errors. The second limitation is Brown and Burton's implicit assumption that there is one correct way to do subtraction. In reality, several different methods are taught, depending upon the curriculum used in the school jurisdiction. Human diagnosticians might also make this assumption; it is undesireable in either case. It must also be noted that BUGGY does not deal with the remediation of subtraction errors; presumably, it is assumed that the detailed identification of the child's procedural errors gives sufficient information for remedial purposes. Despite these limitations, BUGGY established that computers can be used to perform detailed diagnoses of educational problems. It demonstrated the value of a hierarchical approach to the organization of diagnostic knowledge, and it can be adapted to any given test of integer subtraction.

13.2 Young and O'Shea

Young and O'Shea (1982) also dealt with the diagnosis of children's subtraction errors. They represented the subtraction algorithms with production rules. Using a much simpler model of subtraction than BUGGY, which contained twenty rules, they were able to account for most of the errors in the examples they investigated.

In this system, the subtraction process is represented with production rules in the *OPS-2* production rule system. Subtraction errors are represented by omitting necessary productions, adding unneeded productions, and by permuting the order of productions. In the interests of keeping the model simple, the amount of modification of a correct set of production rules was limited by drawing the rules from a small "kit" of commonly used rules, and by making no explicit mention of any number except zero in these rules.

Young and O'Shea were struck by the large number of primitive bugs used in BUGGY, and by the minor nature of the differences between many of these primitive bugs. Their model was designed to use fewer, more general rules to explain subtraction errors; that is, it had a larger *grain size*. After analyzing a corpus of 1570 subtraction problems done by thirty-three ten year old schoolchildren, the production rule representation was able to account for 128 of 170 errors. Two of the children were found to have corrected some of their errors after the first occurance of the error; when this inconsistency was allowed for, the system was able to account for 150 of the 170 errors.

These researchers also recognized that there were different methods of doing subtraction. They explained the decomposition method and the equal addition method, which are the two techniques most widely taught in the United Kingdom, and incorporated enough flexibility into their model to account for this. This flexibility, and the larger grain size of the production rule model, result in a system that is superior to BUGGY in the areas of generality and explanatory power. However, like BUGGY, Young and O'Shea's system does not deal with remediation.

1.3.3 Jones and McLeod

The Jones and McLeod expert system (Colbourn 1982) is designed to diagnose reading difficulties. It is the first system implemented in the Computer-Guided Diagnosis project, of which the expert system described in this thesis is also a part. The Jones and McLeod is based on the McLeod educational diagnostic model, and is written in Lisp using production rules in a traditional expert system style. The performance of the system compares very well with that of expert diagnosticians.

The Jones and McLeod system is based on traditional expert system technology. Information about the case at hand is stored in a database. The database stores such information as the child's name and age, behaviour, motor skills, medical problems, reading and language skills, areas needing further assessment, and so on. Knowledge of diagnostic procedure is encoded in production rules, which are written in Lisp. The program passes through the four phases of the McLeod model as it forms its diagnosis of the child's reading difficulty. When the program finishes the diagnosis, it prints out a report describing the conclusions it has reached and the areas requiring further assessment.

The system's performance on twenty-four test cases was compared with that of professional diagnosticians. The diagnosticians' reports were encoded in the same format as that of the expert system for comparison purposes. Each report was encoded independently by two Special Education students. The variability of the interpretations of the human diagnosticians' reports made consistent comparison with the expert system's results difficult; however, the results were generally good. The expert system was shown to have many benefits: it was much more consistent than human experts; it did not make careless errors that humans sometimes make, such as recording some area as a strength when it was really a relative strength; and it saved time in areas requiring tedious work, such as the detection of error patterns or very long numerical calculations. It established that expert systems are capable of reliable, valid diagnoses.

The Jones and McLeod system is important because it successfully performs diagnoses in the area of reading difficulties, which is more complex than the area of subtraction, and because it is based upon a diagnostic model with firm educational foundations. It is a major influence on the system described in this thesis.

13.4 McDermott

The *McDermott Multidimensional Assessment of Children* (M-MAC) (McDermott 1985) is a program that is intended to classify children according to intelligence, achievement, adaptive behaviour, and social-emotional adjustment, and to design remediative programs based on its findings. It is an actuarially based program; that is, it bases its conclusions on numerical psychometric measurements. It provides the diagnostician with good classification results and some basic remedial conclusions. Its weaknesses are the inherent limitations of the actuarial approach, and the over-generality of the behavioural objectives it specifies for remediation.

M-MAC has two major purposes: classification and program design (remediation). Its classification phase deals with four dimensions of assessment: intellectual functioning, academic achievement, adaptive behaviour, and social-emotional adjustment. Using the results of standardized psychometric tests (which give numerical measurements of mental qualities based on rigourous statistical methods), M-MAC prepares extensive tables, summaries and reports. It also allows comparisons from different observers, such as a teacher and a parent, to be taken into account. In the program design phase, the user selects skill areas and the number of objectives wanted, and the system produces a set of behavioural objectives in a document called an Individualized Educational Program (IEP).

The M-MAC is specially designed to suit the American market, where children's special educational needs are rigidly defined by a federal law that also requires educators to devise IEP's for children with such needs. It classifies children and produces an IEP that is adequate to satisfy legal requirements. The user of the M-MAC can customize the system to suit local requirements and the diagnostic instruments at hand (provided these instruments are in the M-MAC's large pre-defined set of allowable tests). Finally, this system can be run on a common Apple II microcomputer with 48 K bytes of memory and a single floppy disk drive; this makes it much more accessible than BUGGY, the Young and O'Shea system, or the Jones and McLeod system, all of which require mainframe computers.

The M-MAC has several weaknesses that are outlined in (McLeod 1986a). The educational program that it produces, although adequate for U.S. legal purposes, does not serve the needs of a special educator. Its statistical classifications can be confusing, even for an experienced diagnostician. For example, given a 19 point difference between Verbal and Performance sections of the WISC-R intelligence test, the user is told that the difference is not uncommon enough to be considered abnormal; yet the user is also told that this difference is statistically significant at the 0.005 level. But the greatest weakness of the M-MAC is its actuarial orientation. It enforces a very strict psychometric interpretation of test results; for example, if a child has been tested on one test just before his or her birthday and is tested on another just after his or her birthday, M-MAC will not allow the results of the two test to be compared, since the child was officially in different age groups for each test. To satisfy the M-MAC, a diagnostician would have to re-test the child. Retesting is not reliable, since there is a "practice effect" if a test is administered twice in a short time. The dependability of the tests upon which the M-MAC bases its conclusions is also a problem; even in the most reliable and highly respected tests, norming samples are often small (as in the widely used and well respected WISC-R, an intelligence test that used only 200 children for norming per age group), and are drawn from different groups for each test.

Despite these weaknesses, the M-MAC is important because it sets a standard for comprehensiveness and because it attempts to use a sound psychometric basis for its conclusions. It also shows that a large diagnosis program can run on a computer with modest capabilities.

CHAPTER 4:

IMPLEMENTATION AND DESIGN DECISIONS

14. SYSTEM COMPONENTS

Our expert system for arithmetic diagnosis is composed of four software components: Simple THEORIST, a parser, a driver program, and a task hierarchy.

- Simple THEORIST is a restricted version of THEORIST that builds theories based on facts and defaults.
- The parser processes input from the user, turning it into a form that is usable by Simple THEORIST.
- The driver program is a Prolog program that invokes Simple THEORIST as needed, and records the progress of the diagnosis. Much of the control information is in this component.
- The task hierarchy is a large data structure containing background information, facts, possible hypotheses, and queries about the arithmetic tasks being analyzed. This is where the expert knowledge is stored.

It is envisioned that the expert systems that will be built by the Computer-Guided Diagnosis project will contain three hierarchies: task, mode of presentation, and mode of response. Each of these hierarchies will contain two components: an object, and either a process (for the task hierarchy), or a channel (for the mode of presentation and mode of response hierarchies). A channel is the communications method used for interaction with the student, such as visually with pictures, or physically by manipulating blocks. The mode hierarchies were not used because they were in the earliest stages of development when the system was being designed, and because they were not necessary as a part of the feasibility study of the THEORIST paradigm. Future versions of the system will include the mode hierarchies.

There are also two hierarchies which are not realized as software constructs but exist as written specifications. These are the object hierarchy, and the process hierarchy.

- The object hierarchy defines the specializations of objects used in arithmetic.
- The process hierarchy defines the specializations of arithmetic processes.

Simple THEORIST was discussed in Chapter 2. The structure and function of the other components will be detailed in the following sections of this chapter. Actual examples and a description of how a diagnosis proceeds will be described in Chapter 5.

14.1 The Parser

The parser, written by R. G. Goebel, was originally designed to read Prolog programs from the standard input and convert them into a sequence of tokens. These tokens represent the basic lexical units of a Prolog program: atoms, variables, parentheses, square brackets, list separators, implication signs, and semicolons. Atoms are represented with a functor; the Prolog atom "alpha" is represented by the token "atom(alpha)". Variables are also represented with functors; the Prolog variable "Beta" is represented by the token "var("Beta")". The other Prolog lexical elements are represented by atoms that begin with a dollar sign. For example, the left parenthesis character, "(", is represented by the token "$LPAREN". A Prolog fact or rule is translated into a list of these tokens. When the Prolog rule "parent(harry Who) <- father(harry Who);" is parsed, it becomes the list "[atom(parent) $LPAREN atom(harry) var("Who") $RPAREN $IF atom(father) $LPAREN atom(harry) var("Who") $RPAREN $END]".

The parser is a versatile and general software tool. It is used whenever the expert system needs input from the user. The system prompts the user for input, then turns control over to the parser. The user types his or her input, which must be a syntactically correct THEORIST statement terminated with a semicolon, and types a carriage return. The parser converts this into a list of tokens, which can now be processed by the expert system.

14.2 The Driver Program

The driver program has several purposes:

- To control the diagnosis by traversing the task hierarchy during each phase;
- To serve as a link between the task hierarchy, the user, and Simple THEORIST; and
- To perform various "housekeeping" tasks, such as keeping a diagnosis record file, maintaining the diagnosis number, and so on.

The driver consists of 948 lines of Prolog code. Details of how it controls the diagnostic process are described in Chapter 5.

14.3 The Hierarchies

In this expert system, the structure of the hypothesis abstraction hierarchy is based upon a hierarchy of elementary arithmetic skills. These skills are characterized by the type of operation and the type of number. The combination of a particular arithmetic operation, or *process*, and a particular type of number, or *object*, is referred to as a *task*. Processes and objects are represented by hierarchical structures existing as written specifications; tasks are represented by a hierarchy that is realized as a software construct.

The process hierarchy is a hierarchy of elementary arithmetic processes. It was used to guide the design of the task hierarchy. In it, the processes of elementary arithmetic are organized according to their degree of specialization. For example, *operations* was on a hierarchical level above *addition*, *subtraction*, *multiplication*, and *division*.

The process hierarchy was designed with considerably more detail than was used in this system. It encompasses areas such as operations, measurement, geometry, graphing, statistics and probability, ratio and percent, and problem solving. Only the portion of the hierarchy under the area of operations was used in the design of the task hierarchy. This hierarchy is illustrated in Figure 3. The process hierarchy is currently being developed to include non-mathematical areas, such as reading.

The object hierarchy is similar to the process hierarchy, except that it concerns the *objects* of elementary arithmetic instead of the *processes*. The list of objects used in elementary mathematics includes sets, whole numbers, fractions, decimals, operators, operands, mathematical concepts, units of measurement, geometric figures, graphs, distributions, ratios, and all their respective specializations.

As with the process hierarchy, only a small fraction of the hierarchy was actually used in the design of the task hierarchy. This was the subtree under *numbers*, which is illustrated in Figure 4. This hierarchy is also undergoing further development in non-mathematical areas, like reading.

The task hierarchy is a hierarchy of elementary arithmetic skills, defined by combinations of processes and objects, that is represented by a Prolog data structure. It is the hypothesis abstraction hierarchy (Jones and Poole 1985) that Simple THEORIST uses. For reasons of modularity, nodes in the task hierarchy contain not only hypotheses but also facts, queries, background information, and links to other nodes in the hierarchy. In the terms of more traditional expert systems, the hypothesis abstraction hierarchy is the knowledge base.

An example of a task in the hierarchy might be the addition of numbers: the process is "addition", and the object is "number". Therefore, the task hierarchy can be viewed as a tree whose root is an "and-node" with two branches: process and object. When implementing this hierarchy, the "and-branches"

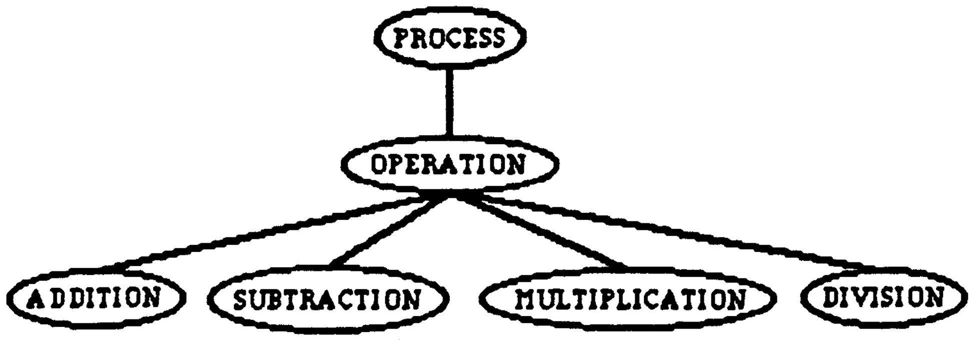


Figure 3. The *process* Hierarchy for Elementary Arithmetic Tasks

have been distributed over the "or-branches"; see Figure 5.

Tasks are most general at the top of the hierarchy, and become more specialized as the levels of the tree increase. The top level of the hierarchy is that of *math-problem;* it encompasses all possible problems with all branches of mathematics. On the level beneath this is *operations,* which is the only specialization of *math-problem* that was implemented in this version. Beneath *operations,* the specializations are differentiated by process; they are *addition-of-numbers, subtraction-of-numbers, multiplication-of-numbers,* and *division-of-numbers.* Directly below this level, tasks are differentiated according to the type of number; under *addition-of-numbers* there are the specializations *addition-of-integers, addition-of-wholes, addition-of-fractions,* and *addition-of-decimals.* The lower levels of the hierarchy are differentiated according to such criteria as: the number of digits involved; the magnitude of the numbers; whether regrouping (borrowing and carrying) is required; and the place value of the positions used for regrouping. A subset of the hierarchy is illustrated in Figure 6.

Embedded within the task hierarchy are facts and defaults used by THEORIST to build its theories. This information, which is also exploited to control the diagnosis, is illustrated in the next few sections.

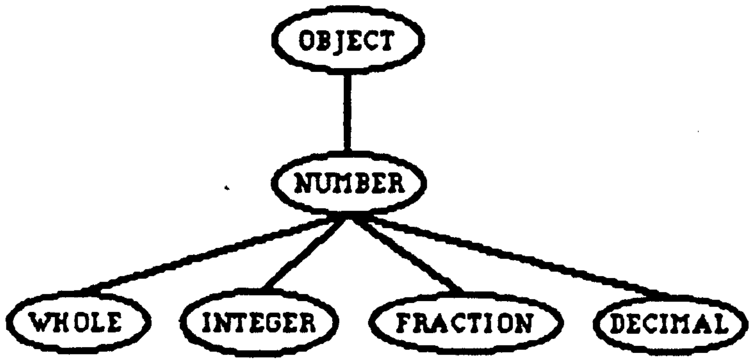


Figure 4. The *object* Hierarchy for Elementary Arithmetic Tasks

An empty node for the task hierarchy is shown in Figure 7. The node is a Prolog functor that encapsulates basic information about the task with THEORIST rules and control information. The names of the fields and their purposes are:

name: A descriptive name of an arithmetic task.

descriptor: Contains the name of the process, a list of input objects, a list containing an output object or objects, and some explanatory text.

generalization: Holds the *name* field of the parent task.

specializations: Holds the *names* of the offspring tasks; these are more specialized versions of the current task.

queries: Contains basic background questions pertaining to the task, which are asked in the Retrospective phase of the diagnosis. Answers to these queries are asserted as facts in the database.

facts: Contains information that is known to be true for this task in any diagnosis. All facts of this type are asserted by Simple THEORIST.

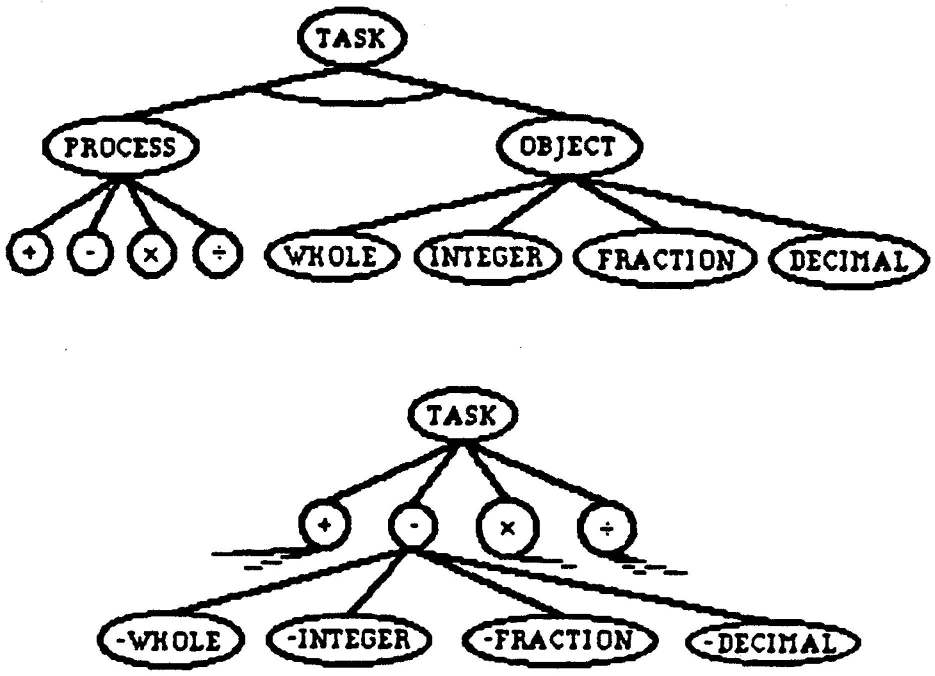


Figure 5. *AND/OR* Tree and Distributed *AND/OR* Tree.

askables: Contains the names of predicates that may be proven by asking the user to supply the relevant information if it is not available from the facts or defaults. Information gained in this manner will also be asserted as facts by Simple THEORIST.

defaults: Contains the hypotheses that can be employed to build the theories that explain the child's problems.

success: A list of names of possible theories that can be used to show a child's mastery of this task. Failure to prove any *success* theory is not considered evidence that a skill has not been mastered.

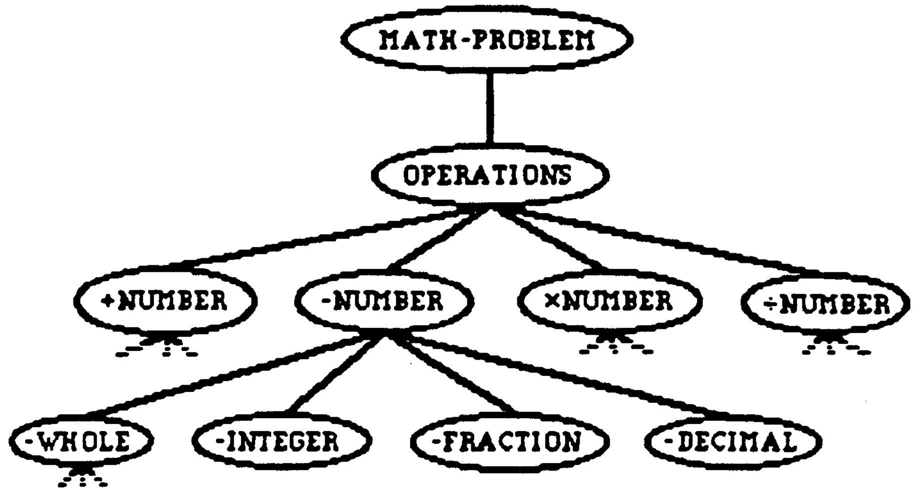


Figure 6. A Subtree of the Task Hierarchy.

bug: This is similar to the *success* field, except that the theories in its list of names are used to demonstrate that the child has *not* mastered a task. Failure to prove a *bug* theory is not considered evidence that the skill has been mastered.

prune: Stores control information; it allows the search to be cut at the appropriate point if the theories formed indicate that this is desirable. The particular nature of the task containing the valid pruning rule determines which task node is the appropriate pruning point.

prescriptions: Contains information showing what corrective measures can be undertaken to remedy a child's problems with this particular task. This information would be exploited during the Prescriptive phase.

The diagnostic expertise stored within the fields *facts, success, bug* and *prune* can all be classified as THEORIST "facts". It is this information that the system must exploit to control the diagnostic process. The fields *facts, askables, defaults, success, bug,* and *prune* may have elements that are only relevant to

```
task(
    name()
    descriptor(
            process-name
            input([])
            output([])
            relevance(""))
    generalization()
    specializations([])
    queries([])
    facts([
            definitive([])
            analytic([])
            ])
    askables([
            definitive([])
            analytic([])
            ])
    defaults([
            definitive([])
            analytic([])
            ])
    success([
            definitive([])
            analytic([])
            ])
    bug([
            definitive([])
            analytic([])
            ])
    prescriptions([])
    prune([
            definitive([])
            analytic([])
            ])
    );
```

Figure 7. An Empty *task* Node

particular phases of the diagnosis; these are indicated accordingly. A detailed example for one particular task will be described in Chapter 5.

1.5. DOMAIN RESTRICTIONS

Most of the effort in designing this expert system went into the code for the Definitive and Analytic phases of the diagnosis. The Retrospective phase was simplified, and the Prescriptive phase was not implemented.

The Retrospective phase of this system is simple, since it was not crucial to the feasibility evaluation of THEORIST. Selected questions from the expert system described in (Colbourn 1982) were used. Research is underway to establish new standards for Retrospective questioning (McLeod 1986b); these standards will be incorporated into future versions.

The system does not deal with the Prescriptive phase of diagnosis. This was deemed to be beyond the scope of this research, which was intended to evaluate the feasibility of using THEORIST, but it must be dealt with in future diagnostic systems. In the Prescriptive phase, such systems would make remedial recommendations based on the theories formed in the Definitive and Analytic phases of the diagnosis.

CHAPTER 5:

HOW A DIAGNOSIS PROCEEDS

16. OVERVIEW

The system carries out a diagnosis by processing the task nodes during four depth-first traversals of the task hierarchy: one traversal for each phase of the diagnosis.

It is useful to view this as a two-level control structure. The hierarchy traversal level controls the overall progress of the diagnosis. The task node processing level deals with the formation of theories about an individual task. Control information processed at the task node processing level can influence the hierarchy traversal level.

To illustrate what happens during a diagnosis, we will examine in detail what the system does as it performs a diagnosis for the task ***subtraction-of-wholes***. The relevant parts of the task hierarchy node for that task will be shown, along with samples of interactions with the system.

17. HIERARCHY TRAVERSAL

The hierarchy traversal level encompasses the entire diagnosis. Each phase of the McLeod model of educational diagnosis has a corresponding traversal of the task hierarchy.

- During the Retrospective phase, the hierarchy is traversed and queries are extracted. These queries are transformed into questions for the user. The answers received from the user are asserted into the database.

- In the Definitive and Analytic phases, facts, defaults, askable questions and possible theories are asserted in the database. Using this information, and the THEORIST inference mechanism, the system forms the theories that constitute the diagnosis and records them. Analytic theories deal with more detailed subskills than Definitive theories and thus are able to be more specific about the child's problems.

- The Prescriptive phase was not implemented.

In each phase, the traversal begins with the root node of the hierarchy, the task *math-problem*. Control passes down the hierarchy, as specified by the *specializations* field of each task node. The traversal is depth-first, with the left-to-right ordering implicitly defined by the order in which the task nodes are listed in the *specializations* fields.

In the Retrospective phase, every node in the hierarchy is examined for possible queries. The design allows for queries to be present in any task node, but all of them are located in the *math-problem* node.

Traversal is more complicated in the Definitive and Analytic phases. After each node is processed, the control structure must determine whether to continue with depth-first searching, or to prune. If pruning rules indicate that pruning must take place at a given node, then the name of the task that is the pruning point is propagated up the hierarchy until control returns to the pruning point task. Processing of that task, and of all specializations beneath it, will be terminated. It should be noted that the pruning mechanism is general enough that it could be implemented for any phase of the diagnosis; it was felt that in this version, pruning during the Definitive and Analytic phases was sufficient.

The Prescriptive phase was not implemented, but in the future, another depth-first traversal of the hierarchy will be made, and a remedial program will be developed, based on the theories formed in the Definitive and Analytic phases.

18. A TRACE THROUGH AN EXAMPLE

The following section traces an sample diagnosis during the Definitive phase for the task *subtraction-of-wholes*. The user's input is in italics.

```
Task = <subtraction-of-wholes>

Trying the SUCCESS version of explain.
Is has-taken-test(bryce,ewrpt-a) true?
[Possible answers: yes; no;
            assume; reply <fact or rule>; help;]
Answer: yes;
Is no-error-in-ewrpt-a-questions-3-or-4 true?
[Possible answers: yes; no;
            assume; reply <fact or rule>; help;]
Answer: no;
Is has-taken-test(bryce,ewrpt-b) true?
[Possible answers: yes; no;
            assume; reply <fact or rule>; help;]
Answer: no;

Trying the FAILURE version of explain.
Proved the FAILURE version of explain.

Trying the SUCCESS version of explain.
Is error-in-ewrpt-a-questions-3-or-4 true?
[Possible answers: yes; no;
            assume; reply <fact or rule>; help;]
Answer: yes;
Proved the SUCCESS version of explain.

Task <subtraction-of-wholes> processed in Phase <definitive>
Success Theories: <[explain-fails]>
Bug Theories: <[problem(subtraction-of-wholes,bryce,ewrpt-a)]>
```

In this example, the system first tries to prove a *success* theory for the task. The first theory it tries depends upon the Enright Wide Range Placement Test, form A. It asks the user if the child, Bryce, has taken this test. The user answers *yes.* It then asks whether the child had no errors in either questions 3 or 4. The user answers *no*, which causes the proof of the the theory to fail. The system then tries the next *success* theory, which depends on form B of the same test. The system asks if Bryce has taken this form of the test, to which the user answers *no*. Since this was the last alternative, all *success* theories had failed, so the system was unable to explain a theory showing that the child had mastered the task *subtraction-of-wholes*.

The next step is to try to prove a *bug* theory, which would mean there is evidence that the child has not mastered the task. The system has already been told which of the Enright tests Bryce has taken, so there is no need to ask again. Instead, it ask the user whether it is true that there was an error questions 3 or 4 on the test. (This information is asked for twice because of problems using the logical *not* in early versions of Simple THEORIST.) The user replies *yes*, and the system is able to prove a bug theory. The system then summarizes what happened during the process of processing this task: *explain* failed to prove a *success* theory, but did prove a *bug* theory that was based upon performance on the Enright Wide Range Placement Test, form A. Finally, this information is recorded in a diagnosis file; this process is not seen by the user.

19. NODE PROCESSING

The node processing level is where theories are formed about a child's performance on the various individual arithmetic tasks. The aggregate of all the theories formed at the node processing level forms the complete diagnosis.

19.1 Phase-Independent Components

Certain components of a task node are relevant during all phases of the diagnosis. For *subtraction-of-wholes*, this information is:

```
name(subtraction-of-wholes)
descriptor(
    subtraction
    input([whole-number,whole-number])
    output([whole-number])
    relevance("The subtraction of whole numbers."))
generalization(subtraction-of-numbers)
specializations([
    sub-whole-1-digit-from-1-digit
    sub-whole-1-digit-from-2-digit
    sub-whole-2-digit-from-2-digit
    sub-whole-3-digit-from-3-digit
    ])
```

The *name* field contains a readable name for the task. The *generalization* field contains the name of the task above this one in the task hierarchy. The tasks immediately below this one in the task hierarchy are held in a list in the *specializations* field. If the task is a leaf node in the task hierarchy, the *specializations* field will be an empty list. The *name, generalization* and *specializations* fields define the structure of the task hierarchy, since each task node is asserted as an independent functor in the Prolog database.

The descriptor is not used in this version of the system but is included for future use. It contains the fields *process*, which gives gives the name of the task's process (in this case, subtraction), taken from the process hierarchy; *input*, which contains a list of input objects, taken from the object hierarchy; *output*, the list of output objects, also from the object hierarchy; and *relevance*, which contains descriptive text about the task.

19.2 Retrospective Components

The purpose of the Retrospective phase is to gather useful background information about the child. In this system, the information is obtained from the user using a simple query mechanism.

In the task hierarchy, a query has the form:

q(*text* [*functor-name description-of-answer*])

For example, a query used for getting the child's last name would have the form:

q("What is the last name of the student?" [name last-name])

This query mechanism works by printing the text that comprises the question, then using the parser to process the user's answer. This is then transformed into a functor and asserted in the database.

Using the example query given above, the process, which includes debugging output, looks like this:

```
What is the last name of the student?
Enter information and end with a semicolon.
What is the value of field <last-name>
(Enter value and end with a semicolon): Harms;
```

The fact *name(Harms)* is then asserted in the database.

There are two main problems with this implementation of the Retrospective phase; both are correctable. The first is that the query mechanism does not perform any error checking on its input. The user might accidently give the child's year of birth as *1799* instead of *1979* and the system will accept it, as well as many other types of errors. The second is that the queries themselves were simple, since they were not crucial to the evaluation of THEORIST. Selected questions from (Colbourn 1982) were used; when work is completed on new standards for Retrospective questioning (McLeod 1986b), the results will be incorporated into future versions.

19.3 Definitive and Analytic Components

The purpose of the Definitive phase is to determine the general area of the problem; the Analytic phase is for determining more precisely what is wrong.

19.3.1 Facts No new facts were defined for this task in these phases. Experience showed that facts were most useful at the top node of the hierarchy (math-problem), and that they were gathered during the Retrospective phase using the queries. They take the form

fact(*functor*)

For example:

```
fact(test-phase(norm-referenced definitive))
```

19.3.2 Defaults Six definitive defaults, four based on the *Enright* test and two on the *KeyMath*, were given.

```
% Enright Wide-Range Placement Test -- form A.

default(problem(subtraction-of-wholes,Student,ewrpt-a)
    [name(Student)
     is-in-grade(Student,Grade)
     ge(Grade,2)
     le(Grade,7)
     has-taken-test(Student,ewrpt-a)
     error-in-ewrpt-a-questions-3-or-4])

% Enright Wide-Range Placement Test -- form B.

default(problem(subtraction-of-wholes,Student,ewrpt-b)
    [name(Student)
     is-in-grade(Student,Grade)
     ge(Grade,2)
     le(Grade,7)
     has-taken-test(Student,ewrpt-b)
     error-in-ewrpt-b-questions-3-or-4])

% KeyMath Test -- Subtraction Part.

default(problem(subtraction-of-wholes,Student,keymath)
    [name(Student)
     has-taken-test(Student,keymath)
     score(Student,keymath-sub-ceiling,Sc)
     lt(Sc,10)])

% Enright Wide-Range Placement Test -- form A.

default(noproblem(subtraction-of-wholes,Student,ewrpt-a)
    [name(Student)
     is-in-grade(Student,Grade)
     ge(Grade,2)
     le(Grade,7)
     has-taken-test(Student,ewrpt-a)
     no-error-in-ewrpt-a-questions-3-or-4])
```

```
  % Enright Wide-Range Placement Test -- form B.

  default(problem(subtraction-of-wholes,Student,ewrpt-b)
      [name(Student)
       is-in-grade(Student,Grade)
       ge(Grade,2)
       le(Grade,7)
       has-taken-test(Student,ewrpt-b)
       no-error-in-ewrpt-a-questions-3-or-4])

% KeyMath Test -- Subtraction Part.

default(noproblem(subtraction-of-wholes,Student,keymath)
    [name(Student)
     has-taken-test(Student,keymath)
     score(Student,keymath-sub-ceiling,Sc)
     ge(Sc,10)])
```

There are also eight Analytic defaults, based on the Enright Skill Placement test and the Enright Basic Facts test. They are very similar to the Definitive rules and are omitted for the sake of brevity; they can be found in Appendix C.

Most rules in the system follow the same kind of pattern. The first rule is a *bug* rule which represents the theory that the student has a problem with subtraction of whole numbers based on evidence from the Enright Wide-Range Placement Test. The conditions necessary for this are given in the functors between the square brackets ("[" and "]"). The conditions are:

- The student's name is in the database. This is needed for the next condition to work correctly.
- The student is in a certain grade in school.
- The grade of the student is greater than or equal to two.
- The grade of the student is less than or equal to seven.
- The student has taken the Enright Wide Range Placement Test, form A.
- Finally, the student had an error in question three or four in this test.

Most of the conditions are safety checks: they make sure that this particular child is in the correct grade range for the test, and that he or she has taken the test. The crucial part of the rule is the last line. The

EWRPT is a test that asks two questions about each area of elementary arithmetic. The questions dealing with the subtraction of whole numbers are questions one and two. An error in either of these questions is considered evidence that there may be a problem with that skill; the diagnostician would then administer a Skill Placement Test on that particular skill to get a more comprehensive assessment of the child's problems with that skill. Theories based on the results of the Skill Placement Tests are used in the Analytic phase.

The other rules are similar. The ones using *ewrpt-b* use the B form of the EWRPT; that form is intended to be given to the child after remedial measures have been taken, so that any improvement can be measured. The rules that are denoted *noproblem* are used for the *success* theories.

19.3.3 Askables The set of *askables* for *subtraction-of-wholes* defines what questions THEORIST is allowed to ask the user in the course of trying to prove a theory. The Definitive askables are:

```
askable(name(S))
askable(is-in-grade(S,G))
askable(has-taken-test(S,ewrpt-a))
askable(has-taken-test(S,ewrpt-b))
askable(error-in-ewrpt-a-questions-3-or-4)
askable(error-in-ewrpt-b-questions-3-or-4)
askable(no-error-in-ewrpt-a-questions-3-or-4)
askable(no-error-in-ewrpt-b-questions-3-or-4)
askable(score(St,T,Sc))
```

The Analytic askables are:

```
askable(name(S))
askable(is-in-grade(S,G))
askable(score(St,T,Sc))
askable(has-taken-test(S,ebft-a-subtraction-of-wholes))
askable(has-taken-test(S,ebft-b-subtraction-of-wholes))
askable(has-taken-test(S,espt-a-subtraction-of-wholes))
askable(has-taken-test(S,espt-b-subtraction-of-wholes))
```

The role of *askables* is explained in context in the section on "How Theories Are Proved".

19.3.4 Possible Theories There are two kinds of theories: *success* theories and *bug* theories. The *success* and *bug* theories are the same for the Definitive and Analytic phases. The *success* theory is:

```
explain(noproblem(subtraction-of-wholes,Student,Test),Theory)
```

The *bug* theory is:

```
explain(problem(subtraction-of-wholes,Student,Test),Theory)
```

Explain is a primitive command of Simple THEORIST. To prove a theory, Simple THEORIST executes the *explain*. If it succeeds, the variables *Student, Test,* and *Theory* become bound to the student's name, the test that was used to prove the theory, and the theory (that is, the list of defaults) itself.

19.3.5 Pruning Rules The pruning rules for *subtraction-of-wholes* cut the search if there is clear evidence that the child has mastered the skill, or if there is no evidence for mastery or non-mastery. The Definitive and Analytic phases both have the same pruning rules, for this particular task. They are:

```
% Cut at "subtraction-of-wholes" if there is no evidence of
% mastery or non-mastery.
prune(    s(explain-fails)
    b(explain-fails)
    subtraction-of-wholes)

% Cut at this point if it is clear that there is evidence
% of mastery and no evidence of non-mastery.
prune(    s(noproblem(_ _ _))
    b(explain-fails)
    subtraction-of-wholes)
```

The first pruning rule means: if there is no evidence of mastery of the task (that is, no success theory can be proved), and there is no evidence of non-mastery (no bug theory was proved), then terminate the search of the task hierarchy at the task *subtraction-of-wholes.* Neither *subtraction-of-wholes* nor any specializations of it will be processed again in this phase.

The second rule is similar, but is activated if any *success* theory was proven and no *bug* theory was proven; cutting takes place at the task *subtraction-of-wholes.*

For this task, pruning will not take place if a bug theory can be proven. It should be noted that pruning rules in a real system would be much more sophisticated than this.

19.4 How Theories Are Proved

A mathematical description of the operations of THEORIST was given in Chapter 2. In this section we will describe how Simple THEORIST operates on actual examples.

Consider the theory

```
explain(problem(subtraction-of-wholes,Student,Test),Theory)
```

and the default

```
% Enright Wide-Range Placement Test -- form A.

default(problem(subtraction-of-wholes,Student,ewrpt-a)
    [name(Student)
     is-in-grade(Student,Grade)
     ge(Grade,2)
     le(Grade,7)
     has-taken-test(Student,ewrpt-a)
     error-in-ewrpt-a-questions-3-or-4])
```

Simple THEORIST tries to prove the theory in the following way: it invokes the *explain* procedure on the query *problem(subtraction-of-wholes,Student,Test)*, with *Student* bound to the child's name. First it checks its database of facts. If a fact matching the query exists in the database, the query will succeed; *Test* will be bound to the name of the test in the fact and *Theory* will be bound to the empty list ("[]"). However, in this system, these theories will not be found in the database of facts.

Instead, the theories will be found in the database of defaults. To prove these defaults, it will be necessary to prove each of the rules in its body. Each of these rules is proved in the same way as the theory itself: first by checking the facts, then by checking the body of defaults, and finally by asking the user, if this has been allowed with an *askable* statement.

19.5 Pruning Rules and Control Flow

Consider the following pruning rule:

```
prune(     s(explain-fails)
    b(problem(_ _ _)
    subtraction-of-numbers)
```

The system uses the rule in the following way: it compares the *success* and *bug* theories it has formed at the current node to the contents of the *s(...)* and *b(...)* functors in the pruning rule. If they match, then it returns *subtraction-of-numbers* as the pruning point; if not, it returns the empty list, "[]".

If *subtraction-of-numbers* is returned as pruning point, it alters the flow of control. In the event that the node currently being processed is *subtraction-of-numbers,* all exploration of its specializations is stopped before it even begins, and control passes to brother nodes of *subtraction-of-numbers* in the hierarchy. When the current node is a specialization of *subtraction-of-wholes,* processing of its specializations is cut off, and the control returns to higher levels of the hierarchy, with the pruning point being propagated upward, cutting off the search until the pruning point is reached.

If there is no pruning point, then a depth-first search of each of the specializations of this task is carried out. The processing of any one of these specializations may produce a pruning point that could propagate back up to this level and above. This process continues until the diagnostic phase is complete.

CHAPTER 6:

TESTING

20. OVERVIEW

Testing of the expert system took place throughout its design and implementation stages; the information yielded in this process helped refine the design and improve performance. The numerous unrecorded test cases showed that the system does indeed work smoothly and that future improvements are needed. For the final test phase, two formal test cases were devised; each one was carefully designed to illustrate many aspects of the system's performance. Three test cases based on actual clinical data were also made available by an arithmetic diagnostician at the University of Saskatchewan. Details of these test cases are found in Appendix B.

The purposes of testing were: to verify that THEORIST, in its Simple THEORIST manifestation, works properly; to evaluate the THEORIST paradigm of theory formation from a hierarchical organization of facts and defaults as a foundation for expert systems development; and to test the driver program for proper tree traversal, correct pruning, input/output, record-keeping, and ease of use. Clinical accuracy of diagnosis was of secondary concern, since the rules and the particular task hierarchy were not devised by educators. What is wanted is a good system framework that can have the details filled in with the help of experts in the future.

The two non-clinical test cases, denoted John and Janelle, are designed to complement each other. The John case represents a child who is having difficulties with all areas of arithmetic. It emphasises pruning at general levels of diagnosis. The Janelle case represents a child who has difficulties with only one area. This case involves exploring almost all of the hierarchy. Both cases demonstrate theory formation hierarchy traversal, pruning, input/output, and record-keeping.

The three clinical cases, Ashley, Sue, and Cam, demonstrated the reliability of the framework; the clinically-derived cases demonstrated what the system would do with information based on actual diagnoses. The subjects had only received the KeyMath test, which is only useful for very general tasks; however, an expert diagnostician extrapolated what their performance would be on the Stanford and some of the Enright tests, based on his intimate familiarity with the subjects. (The diagnostician was not familiar with THEORIST and hence was not biased in his extrapolations.) The cases showed results between those of John and Janelle; they gave results that had more depth than those of John, but because not all of the Enright test information was available, they were not as detailed as those of Janelle, particularly in the tasks relating to the subtraction of whole numbers. Experience with clinical test cases also suggested certain improvements to the pruning rules. As the domain expertise was not drawn directly from experts, it was not surprising that improvements in pruning rules were needed; such improvements will be incorporated into future versions.

21. WHAT TESTING REVEALED

Theory formation worked properly. For example, in the John case, when Simple THEORIST was asked to *explain* the theory *problem(addition-of-wholes,john,Test)*, the system selected an appropriate possible theory, *problem(addition-of-wholes,john,ewrpt-a)*, asked the user what John's grade was, whether he had taken the test, and whether he had an error in questions one or two. The theory *problem(addition-of-wholes,john,ewrpt-a)* was returned. No problems with theory formation were encountered.

It had been predicted by experts in the field of educational diagnosis that the hierarchical organization of hypotheses would prove to be a good idea (Computer-Guided Diagnosis Workshop 1985); this was confirmed by testing. In the case of John, the tasks *math-problem, operations, addition-of-numbers,* *addition-of-wholes, subtraction-of-numbers, subtraction-of-wholes, multiplication-of-numbers, multiplication-of-wholes, division-of-numbers,* and *division-of-wholes* were explored, with hierarchy traversal pruned at the level of **<operation>***-of-wholes*. When this is contrasted with the Janelle case, which explored the hierarchy to much greater depth, it is clear that the hierarchical organization of theories can reduce search time considerably. Time savings will be even greater in future diagnostic systems with much larger hierarchies.

Hierarchy traversal took place properly. A detailed transcript illustrating this is found in Appendix A, since it is too long to show here. The pruning mechanism also worked well, as was indicated in the John case. Pruning information was propagated up the tree to the correct level, cutting off search at that point.

Testing also revealed that ease of use and error correction need improvement. For example, if the user is to input factual information, such as Janelle's grade, he or she must do it in the form:

```
reply fact(is-in-grade(janelle,3));
```

The syntax must be exactly right, but the information can be completely wrong. As there is no error checking, one could tell the system that Janelle is in grade 1024, and it would be accepted. In the future, a user interface must be developed, to ease interaction with the system and screen the input for errors.

Finally, record keeping worked well. Each individual session has its own diagnosis file. For the Janelle case, the file begins:

```
retro(name(janelle));
retro(birth-year(1978));
retro(birth-month(11));
retro(birth-day(5));
retro(referral-year(1985));
retro(referral-month(10));
retro(referral-day(4));
retro(report-year(1985));
retro(report-month(11));
retro(report-day(21));
```

For each task that gets processed in the hierarchy, the system makes a record like this:

```
record(
    diagnosis-number(12)
    name(janelle)
    task-name(math-problem)
    phase(definitive)
    s-theory([explain-fails])
    b-theory([problem(math-problem,janelle,sdmt-red)])
    );
```

A problem that became evident after working with the system is that the control structure is too rigid. The driver program always performs the same actions for each task node. It may be the case that, in more complex domains, different things should be done for different tasks. The M-MAC indicates that there are some situations (such as the classification of children for funding purposes, according to legally defined standards) where actuarial techniques, rather than heuristic ones, may be more desirable. More flexibility should be allowed in future versions, such as the use of actuarial techniques and the incorporation of error-analysis routines.

CHAPTER 7:

FUTURE CONSIDERATIONS AND CONCLUSIONS

The intent of this thesis was to determine whether the THEORIST paradigm is an appropriate foundation for the development of expert systems for educational diagnosis. After the design, implementation, and testing of the small expert system for the diagnosis of children's arithmetic difficulties, we have concluded that the THEORIST paradigm is appropriate for such systems. We have also reached conclusions regarding the strengths and weaknesses of this particular implementation of THEORIST, and of the expert system design that supports it. The experience of implementing this version has also suggested improvements for future educational diagnostic expert systems.

The THEORIST paradigm proved to be well suited to educational diagnosis, as experts in the domain had predicted (Computer-Guided Diagnosis Workshop 1985). Reasoning from defaults as well as facts allows one to model the uncertainties and inconsistencies that exist in the domain, such as the presence of evidence for and against the mastery of a skill. The presence of these inconsistencies leads the system to deeper analysis, to determine the most specific explanation of the problem; the result of this process is a deep model of the child's problems. THEORIST's default reasoning system also indicates what portions of the resulting theories are based on fact, and what portions are based on assumptions; each default used in the construction of a theory is returned as a component of the resulting theory, whereas the facts used in the construction are not. The hierarchical organization of facts and hypotheses kept small the number of possible theories to be considered during a diagnosis of any given task; this considerably reduced the solution space for each task in the hierarchy. The search pruning mechanism increased the speed of the diagnosis and kept the diagnosis focussed on the relevant aspects. These central ideas will provide a sound foundation for future educational diagnostic expert systems.

In any prototype, one expects to find weaknesses,and the expert system described in this thesis is no exception. The most prominent weakness is that the hierarchies, theories, and pruning rules were not designed by educators (although input was received from educators and did influence the design). This weakness is not crucial, since the goal of the research was the evaluation of the THEORIST paradigm, not clinical validity. As a part of the Computer-Guided Diagnosis project, design of the system's knowledge base, including the hierarchies, theories, and pruning rules, is being carried out in cooperation with educators.

Another shortcoming resulted from the use of Simple THEORIST rather than complete THEORIST. When Simple THEORIST is asked to explain an observation, it will halt after it finds the first theory that explains the observation. To obtain alternate theories, the user must explicitly prompt Simple THEORIST. What is needed is *all* the provable theories regarding a given observation. However, even if these theories were available, no theory comparitor, which would evaluate the resulting theories and choose the best one, has yet been implemented. Simple THEORIST is only an experimental tool (and is the only version of THEORIST currently available) and hence has not had these features implemented. Future versions of THEORIST will generate all valid theories and will compare them to determine the best one (where "best" is defined according to criteria like those specified in (Poole 1985)).

The rigidity of the control structure is another weakness of the current design. Every task node is processed in the same way; no allowance is made for performing specialized actions that may be unique to a given node, such as performing an error analysis. Proposed corrections to the problem are explained in detail in the section on Future Considerations.

Finally, the user interface is very poor. It demands that the user adhere to its syntactic conventions, and it does not check input for errors. An improved interface must be designed in cooperation with the users themselves; this is discussed further in the Future Considerations section.

22. *FUTURE CONSIDERATIONS*

The lessons learned through the experience of building the expert system described in this thesis have suggested improvements and enhancements that could be incorporated into future versions. The most obvious such improvement is to build the system's hierarchies, rules, and pruning rules from the specifications of the domain experts, the educational diagnosticians. This was not needed in a system whose main concern was not clinical validity, but needless to say, it is essential for any expert system if it is to be accepted by its intended users. The Computer-Guided Diagnosis project, of which this research is a part, is currently eliciting expertise from a large body of diagnosticians from across Canada, and will incorporate the knowledge it gains in this manner into future versions of the expert system.

As a part of this process of incorporating legitimate expertise into future expert systems, it will be necessary to revise the existing task hierarchy, and to incorporate hierarchies for mode of presentation and mode of response. These hierarchies will put the diagnoses formed by the system on a firmer educational basis.

A new user interface for the system is highly desirable. The current one is very rudimentary, as it is based on a parser intended for reading Prolog programs; also, it has no error-checking capability. The new interface must make the system easy to use, and provide thorough error checking of the user's input. Two user interface models are being considered: a window, menu, and mouse interface, like that popularized by the Apple Macintosh; and a natural language interface, like that discussed in (Kindersley 1986). Good natural language interfaces are hard to build and occupy large amounts of memory and storage space. This must be balanced against the goal of delivering a system that can run on commonly available school microcomputers, which tend to have limited main memory and secondary storage. In such situations, the window, menu, and mouse type of user interface may be more practical. Consultation with experts in the area of Human Factors would be helpful when future versions of the user interface are being designed.

The flexibility of future systems must be improved, particularly at the node processing level. Currently, during a given phase, the same procedure is followed for the processing of every node: in the Retrospective phase, the queries are extracted and presented to the user; in the Definitive and Analytic phases, facts, defaults, and askables are added to the database, Simple THEORIST is invoked on possible theories, and the pruning rules are executed. It seems desirable to allow in task nodes program code that can override these default execution sequences in the appropriate circumstances. For example, in certain situations, like that of classification of children for funding purposes, it might be better to supplement the heuristic theory-building techniques of THEORIST with actuarial techniques (such as those employed by the M-MAC). The incorporation of powerful error analysis routines, like those that BUGGY tried to provide, is another situation calling for the freedom to incorporate additional program code when necessary. Such a capability would not be difficult to add to the existing THEORIST-based system and would increase the effectiveness of the system.

The implementation of the pruning rules should be improved in future. Presently, the program does a simple pattern match on the success and bug theories formed during the processing of the task node. If the match succeeds, the pruning rule yields a corresponding pruning point, which is propagated up the hierarchy until the pruning point is reached. (Implementation of pruning point propagation proved to be a surprisingly difficult piece of programming.) All processing of the task hierarchy from the pruning point on down is terminated for the remainder of that phase.

This implementation of pruning is simplistic in a number of respects. First, it depends solely on the theories formed during the processing of the task node that contains the pruning rule; no reference is made to previously formed theories, or to any other available information. Secondly, pruning information is forgotten once the pruning has occurred. Thirdly, only one pruning point can be active at a given time. Lastly, the pruning rules have not been designed in cooperation with experts.

The first problem with pruning rules can be solved by abandoning simple pattern matching and replacing it by writing each pruning rule like a program subroutine, so that it is able to base the pruning decision on other conditions as well as on the basis of theories formed for the current task node. The second and third problems can be dealt with by keeping the pruning points that the pruning rules produce in a data structure that would last for the duration of the diagnosis. Pruning points would be placed in this structure during the course of the execution of the pruning rules. With such a structure in place, it would then become reasonable to allow the pruning rules to yield more than one pruning point, and to allow "de-pruning"; that is, the deletion of pruning points from the structure if it becomes clear from context that some tasks should be investigated after all. Finally, the problem of pruning rules made without expert advice will be solved at the same time the same problem with the hierarchies and rules is dealt with.

An area where further research should be conducted before future versions of THEORIST are built is that of the necessity of consistency checking of theories. This was motivated by theoretical and empirical factors. The theoretical argument against consistency checking is that, in the worst case, the consistency check is not guaranteed to terminate. The empirical argument is that consistency checking incurs a computational overhead, and that the theories formed for this system were simple and could not imply the negations of any of the defaults used in their proofs. This empirical argument may well change when rules are written after consultation with experts. However, once such rules are in place, research should be conducted as to whether consistency checking is needed and whether it will terminate, given the nature of the rules devised by educators. If it is found to be unnecessary, it eliminates both an unneeded computational overhead for the typical case, and the possibility of a non-terminating check in the pathological case.

A logical successor to the expert system this thesis has described would be a THEORIST-based expert system for a domain more complicated than elementary arithmetic, such as reading. A prototype expert system for this domain has already been built (Colbourn 1982), using traditional expert systems techniques, so the feasibility of such a system has been established. To implement such a system in THEORIST, it would be necessary to transform its "flat" production rule structure into an appropriate hypothesis abstraction hierarchy, and to turn the rules themselves into THEORIST facts and defaults. Much useful knowledge, especially of hierarchy design, would be gained in this exercise.

In the longer term, certain other ideas may prove useful. One such idea is that of devising diagnostic tests that are specifically designed for use in computer-guided diagnosis. Current diagnostic tests must be limited in size and scope so that they can be administered, scored, and interpreted in a reasonable amount of time. Some tests now in use, such as the *Stanford Diagnostic Mathematics Test*, are so complex that they offer optional computerized scoring and interpretation. A computer-guided diagnostic test could have an extremely large number of components, an appropriate subset of which would be chosen to be administered for the case at hand. Computerized scoring and interpretation would be available on site, increasing the speed and accuracy of test interpretation. The results of earlier sets of subtests could be used to guide the selection of further subtests, which would focus sharply on the problem areas. Error analysis routines, like those of BUGGY, could be available for analyzing test results, further increasing the useful diagnostic information yielded by the tests. The devising of such tests would undoubtedly be a major undertaking, and will probably not occur until the benefits of computer-guided diagnosis become widely known amongst special educators.

Finally, the ultimate goal of this work will be the construction of a computer-guided diagnosis information system, of which the expert system will be one important part. Such a system would allow the diagnostician to keep ongoing records of diagnoses of many students. Diagnosis would be guided by the expert system component. A database of diagnostic tests would supply information about the tests, similar to (Jones and Loken 1985). Computer-guided diagnosis tests would be available on-line, as well as facilities for their scoring and interpretation. A report generator would put the information gathered and the conclusions reached into a useful, readable report format. Users would have an editing capability available, so that they could incorporate information about tests and other data that the system may not know about, and so that past erroneous entries can be corrected. Such a system should not be built until more experience has been accumulated from future versions of the expert system, and feedback is received from the users regarding what they expect from such a system. A deliverable system must also wait for the introduction of microcomputers with the necessary speed, memory, and secondary storage to store and execute this large system. When such a system becomes available, educational diagnosticians will have a tool of unparallelled power available to them.

The experience gained in the construction of the expert system that has been described in this thesis has been invaluable. The ideas for the improvement of the system will be incorporated into future versions. The promise of expert systems for educational diagnosis will soon be fulfilled.

REFERENCES

Barr, A., and Feigenbaum, E. A. (Eds.). (1982) *The Handbook of Artificial Intelligence*. Los Altos: William Kaufmann, Inc.

Buchanan, B. G., and Shortliffe, E. H. (Eds.). (1984) *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Reading, Mass.: Addison-Wesley.

Brown, J. S., and Burton, R. R. (1978) "Diagnostic Models for Procedural Bugs in Basic Mathematical Skills". *Cognitive Science* 2. Pages 155—192.

Charniak, E., and Wilks, Y. A. (1976) *Computational Semantics: An Introduction to Artificial Intelligence and Natural Language Comprehension*. Amsterdam: North-Holland.

Colbourn (Jones), M. L. (1982) *Computer-Guided Diagnosis of Learning Disabilities: A Prototype*. M.Ed. Thesis, University of Saskatchewan.

Colbourn (Jones), M., and McLeod, J. (1983) "The Potential and Feasibility of Computer-Guided Educational Diagnosis", *Proceedings of the International Federation for Information Processing (IFIP) Conference*, 891—896. Paris, France.

Computer-Guided Diagnosis Workshop (1985). Banff, Alberta.

Davis, R. (1976) *Applications of Meta-Level Knowledge to the Construction, Maintenance, and Use of Large Knowledge Bases*. Doctoral dissertation, Stanford University.

Hammill, D. D., and Bartel, N. R. (1982) *Teaching Children with Learning and Behavior Problems* (third edition). Boston: Allyn and Bacon.

Harmon, P. and King, D. (1985) *Expert Systems: Artificial Intelligence in Business*. New York: Wiley Press.

Hayes-Roth, F., Waterman, D. A., and Lenat, D. B. (1983) *Building Expert Systems*. Reading: Addison-Wesley.

Human Edge Software. (1984) *Expert-Ease User Manual*. Palo Alto: Human Edge Software Corporation.

Jones, M. (ed.) (1986) *AI Approaches to Education*. Special issue of *Computational Intelligence*. (In press.)

Jones, M. and Loken, M. (1985) *A Database of Psycho-Educational Tests*. Department of Computer Science Research Report CS-85-17, University of Waterloo.

Jones, M. and Poole, D. (1985) "An Expert System for Educational Diagnosis Based on Default Logic", *Proceedings of the Fifth International Conference on Expert Systems and Their Applications*, 673—683. Avignon, France.

Kindersley, S. M. (1986) *A Natural Language Interface for an Educational Diagnosis Expert System*. M.Math. Thesis, University of Waterloo, (in preparation).

Lederberg, J. (1964) *Computation of Molecular Formulas for Mass Spectrometry*. San Francisco: Holden-Day.

Lindsay, R., Buchanan, B. G., Feigenbaum, E. A., and Lederberg, J. (1980) *DENDRAL*. New York: McGraw-Hill.

McDermott, P. A. (1985) *M-MAC: McDermott Multidimensional Assessment of Children.* Cleveland: Psychological Corporation.

McLeod, J. (1982) "Institute of Child Guidance and Development at the University of Saskatchewan", *Journal of Learning Disabilities*, vol. 15, no. 5, pages 290—293.

McLeod, J. (1986a) *Review of M-MAC.* Personal communication.

McLeod, J. (1986b) "Saskatoon Diagnostician's Group: Brainstorming Session". University of Saskatchewan: Computer-Guided Diagnosis Project memorandum.

Poole, D. L. (1982) *The Theory of CES: A Complete Expert System.* Ph.D. Dissertation, Department of Computer Science, Australian National University.

Poole, D. L. (1984) *The Logical Definition of Deduction Systems.* Department of Computer Science Research Report CS-84-12, University of Waterloo.

Poole, D. L. (1985) "On the Comparison of Theories: Preferring the Most Specific Explanation", *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 144—147. Los Angeles.

Poole, D. L., Aleliunas, R., and Goebel, R. (1986) *THEORIST: a Logical Reasoning System for Default and Diagnosis.* Department of Computer Science Technical Report CS-86-06, University of Waterloo.

Pople, H. E., Jr. (1977) "The Formation of Composite Hypotheses in Diagnostic Problem Solving — An Exercise in Synthetic Reasoning". *IJCAI 5*, pages 1030—1037.

Popper, K. R. (1959) *The Logic of Scientific Discovery.* New York: Basic Books.

Quine, W. V. and Ullian, J. S. (1978) *The Web of Belief*, second edition. New York: Random House.

Sacerdoti, E. D. (1977) *A Structure for Plans and Behavior.* New York: American Elsevier.

Schank, R. C., and Colby, K. (1983) *Computer Models of Thought and Language.* San Francisco: Freeman.

Shortliffe, E. H. (1976) *Computer-Based Medical Consultants: MYCIN.* New York: American Elsevier.

Sleeman, D., and Brown, J. S. (eds.) (1982) *Intelligent Tutoring Systems.* London: Academic Press.

Wallace, G., and Larsen, S. C. (1979) *Educational Assessment of Learning Problems: Testing for Teaching.* Boston: Allyn and Bacon, Inc.

Waltz, D. L. (1978) "An English Language Question-answering System for a Large Relational Data Base," *Communications of the ACM*, Vol. 21, No. 7.

Winston, P. H., and Prendergast, K. A. (Eds.). (1984) *The AI Business.* Cambridge, Mass.: The MIT Press.

Young, R. M., and O'Shea, T. (1982) "Errors in Children's Subtraction", *Cognitive Science*, II, 153—177.

APPENDIX A

A SAMPLE SESSION WITH THE EXPERT SYSTEM

This appendix contains a script of an interactive session with the expert system on the test case John.

Much of the debugging output has been retained, to clarify what is happening.

```
Script started on Thu Jan 23 13:54:51 1986
dragon[1]: Yes Master? wup -r 50 .
Waterloo Unix Prolog [Release 2.0 -- Oct., 1985]
yes

?     more("/u/jbtubman/thesis/st2/.inithelp");
SIMPLE THEORIST TWO - COMPILER

all complaints to dlpoole@water.
Welcome to the compiler, type "help;" for help
good luck
yes
?consult(.test4);

?     consult(taskhier);
yes
yes
?doit;


Diagnosis Begins


Retrospective Phase


Task = <math-problem>


Enter the child's name.
Enter information and end with a semicolon.
What is the value of field <first-name>
(Enter value and end with a semicolon): john;
Asserted <name(john)> in mod.

Enter the year of the child's birth.
Enter information and end with a semicolon.
What is the value of field <year-of-birth>
(Enter value and end with a semicolon): 1979;
Asserted <birth-year(1979)> in mod.

Enter the month of the child's birth (as a number).
Enter information and end with a semicolon.
What is the value of field <month-of-birth>
(Enter value and end with a semicolon): 4;
Asserted <birth-month(4)> in mod.

Enter the day of the child's birth (as a number).
Enter information and end with a semicolon.
What is the value of field <day-of-birth>
(Enter value and end with a semicolon): 1;
Asserted <birth-day(1)> in mod.

Enter the date on which the child was referred (as numbers).
Year:
Enter information and end with a semicolon.
What is the value of field <year-of-referral>
(Enter value and end with a semicolon): 1985;
Asserted <referral-year(1985)> in mod.

Month:
Enter information and end with a semicolon.
What is the value of field <month-of-referral>
(Enter value and end with a semicolon): 10;
Asserted <referral-month(10)> in mod.

Day:
Enter information and end with a semicolon.
What is the value of field <day-of-referral>
(Enter value and end with a semicolon): 2;
Asserted <referral-day(2)> in mod.

Enter the date on which the school report was completed (as numbers).
Year:
Enter information and end with a semicolon.
What is the value of field <year-of-report>
(Enter value and end with a semicolon): 1985;
Asserted <report-year(1985)> in mod.

Month:
Enter information and end with a semicolon.
What is the value of field <month-of-report>
(Enter value and end with a semicolon): 11;
Asserted <report-month(11)> in mod.

Day:
Enter information and end with a semicolon.
What is the value of field <day-of-report>
(Enter value and end with a semicolon): 13;
Asserted <report-day(13)> in mod.
```

Definitive Phase

```
Task = <math-problem>
```

```
Trying the SUCCESS version of explain.
Is is-in-grade(john,_16222) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: reply is-in-grade(john,2);
Is has-taken-test(john,sdmt-red) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: yes;
Is total-stanine-score-in-test(john,sdmt-red,_17078) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: reply total-stanine-score-in-test(john,sdmt-red,2);

Trying the FAILURE version of explain.
Proved the FAILURE version of explain.

Trying the SUCCESS version of explain.
Proved the SUCCESS version of explain.

Task <math-problem> processed in Phase <definitive>
Success Theories: <[explain-fails]>
Bug Theories: <[problem(math-problem,john,sdmt-red)]>


Task = <operations>


Trying the SUCCESS version of explain.
Is stanine-score-in-computation-subtest(john,sdmt-red-computation,_20637) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: reply stanine-score-in-computation-subtest(john,sdmt-red-computation,2);

Trying the FAILURE version of explain.
Proved the FAILURE version of explain.

Trying the SUCCESS version of explain.
Proved the SUCCESS version of explain.

Task <operations> processed in Phase <definitive>
Success Theories: <[explain-fails]>
Bug Theories: <[problem(operations,john,sdmt-red)]>


Task = <addition-of-numbers>


Task <addition-of-numbers> processed in Phase <definitive>
Success Theories: <[]>
Bug Theories: <[]>


Task = <addition-of-wholes>
```

```
Trying the SUCCESS version of explain.
Is has-taken-test(john,ewrpt-a) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: yes;
Is no-error-in-ewrpt-a-questions-1-or-2 true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: no;
Is has-taken-test(john,ewrpt-b) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: no;

Trying the FAILURE version of explain.
Proved the FAILURE version of explain.

Trying the SUCCESS version of explain.
Is error-in-ewrpt-a-questions-1-or-2 true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: yes;
Proved the SUCCESS version of explain.

Task <addition-of-wholes> processed in Phase <definitive>
Success Theories: <[explain-fails]>
Bug Theories: <[problem(addition-of-wholes,john,ewrpt-a)]>

Prune at point <addition-of-numbers>


Task = <subtraction-of-numbers>


Task <subtraction-of-numbers> processed in Phase <definitive>
Success Theories: <[]>
Bug Theories: <[]>



Task = <subtraction-of-wholes>


Trying the SUCCESS version of explain.
Is no-error-in-ewrpt-a-questions-3-or-4 true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: no;
Is has-taken-test(john,keymath) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: yes;
Is score(john,keymath-sub-ceiling,_18443) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: reply score(john,keymath-sub-ceiling,7);

Trying the FAILURE version of explain.
Proved the FAILURE version of explain.

Trying the SUCCESS version of explain.
Proved the SUCCESS verions of explain.
Proved the SUCCESS version of explain.

Task <subtraction-of-decimals> processed in Phase <definitive>
Success Theories: <[explain-fails]>
Bug Theories: <[problem(subtraction-of-decimals,john,ewrpt-a)]>



Task <multiplication-of-numbers> processed in Phase <definitive>
Success Theories: <[]>
Bug Theories: <[]>



Task = <multiplication-of-wholes>


Trying the SUCCESS version of explain.
Is no-error-in-ewrpt-a-questions-5-or-6 true;
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: no;

Trying the FAILURE version of explain.
Proved the FAILURE version of explain.

Trying the SUCCESS version of explain.
Is error-in-ewrpt-a-questions-5-or-6 true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: yes;
Proved the SUCCESS version of explain.

Task <multiplication-of-wholes> processed in Phase <definitive>
Success Theories: <[explain-fails]>
Bug Theories: <[problem(multiplication-of-wholes,john,ewrpt-a)]>

Prune at point <multiplication-of-numbers>



Task = <division-of-numbers>


Task <division-of-numbers> processed in Phase <definitive>
Success Theories: <[]>
Bug Theories: <[]>



Task = <division-of-wholes>


Trying the SUCCESS version of explain.
Is no-error-in-ewrpt-a-questions-7-or-8 true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: no;
```

```
Task <subtraction-of-wholes> processed in Phase <definitive>
Success Theories: <[explain-fails]>
Bug Theories: <[problem(subtraction-of-wholes,john,ewrpt-a)]>


Task = <subtraction-of-integers>


Task <subtraction-of-integers> processed in Phase <definitive>
Success Theories: <[]>
Bug Theories: <[]>


Task = <subtraction-of-fractions>


Trying the SUCCESS version of explain.
Is no-error-in-ewrpt-a-questions-13-or-14 true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: no;

Trying the FAILURE version of explain.
Proved the FAILURE version of explain.

Trying the SUCCESS version of explain.
Is error-in-ewrpt-a-questions-13-or-14 true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: yes;
Proved the SUCCESS version of explain.

Task <subtraction-of-fractions> processed in Phase <definitive>
Success Theories: <[explain-fails]>
Bug Theories: <[problem(subtraction-of-fractions,john,ewrpt-a)]>

Prune at point <subtraction-of-fractions>


Task = <subtraction-of-decimals>


Trying the SUCCESS version of explain.
Is no-error-in-ewrpt-a-questions-21-or-22 true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: no;

Trying the FAILURE version of explain.
Proved the FAILURE version of explain.

Trying the SUCCESS version of explain.
Is error-in-ewrpt-a-questions-21-or-22 true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: yes;
```

```
Is score(john,keymath-div-ceiling,_21680) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: reply score(john,keymath-div-ceiling,2);

Trying the FAILURE version of explain.
Proved the FAILURE version of explain.

Trying the SUCCESS version of explain.
Is error-in-ewrpt-a-questions-7-or-8 true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: yes;
Proved the SUCCESS version of explain.

Task <division-of-wholes> processed in Phase <definitive>
Success Theories: <[explain-fails]>
Bug Theories: <[problem(division-of-wholes,john,ewrpt-a)]>

Prune at point <division-of-numbers>


Analytic Phase



Task = <math-problem>


Task <math-problem> processed in Phase <analytic>
Success Theories: <[]>
Bug Theories: <[]>



Task = <operations>


Task <operations> processed in Phase <analytic>
Success Theories: <[]>
Bug Theories: <[]>



Task = <addition-of-numbers>


Task <addition-of-numbers> processed in Phase <analytic>
Success Theories: <[]>
Bug Theories: <[]>



Task = <addition-of-wholes>


Trying the SUCCESS version of explain.
```

```
Is has-taken-test(john,espt-a-addition-of-wholes) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: yes;
Is score(john,espt-a-addition-of-wholes,_25562) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: reply score(john,espt-a-addition-of-wholes,5);
Is has-taken-test(john,espt-b-addition-of-wholes) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: no;
Is has-taken-test(john,ebft-a-addition-of-wholes) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: yes;
Is score(john,ebft-a-addition-of-wholes,_27970) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: reply score(john,ebft-a-addition-of-wholes,40);
Is has-taken-test(john,ebft-b-addition-of-wholes) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: no;

Trying the FAILURE version of explain.
Proved the FAILURE version of explain.

Trying the SUCCESS version of explain.
Proved the SUCCESS version of explain.

Task <addition-of-wholes> processed in Phase <analytic>
Success Theories: <[explain-fails]>
Bug Theories: <[problem(addition-of-wholes,john,espt-a)]>

Prune at point <addition-of-numbers>


Task = <subtraction-of-numbers>


Task <subtraction-of-numbers> processed in Phase <analytic>
Success Theories: <[]>
Bug Theories: <[]>


Task = <subtraction-of-wholes>


Is has-taken-test(john,espt-a-subtraction-of-wholes) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: yes;
Is score(john,espt-a-subtraction-of-wholes,_29090) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: reply score(john,espt-a-subtraction-of-wholes,9);
Is has-taken-test(john,espt-b-subtraction-of-wholes) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: no;
Is has-taken-test(john,ebft-a-subtraction-of-wholes) true?
```

```
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: yes;
Is score(john,ebft-a-subtraction-of-wholes,_30115) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: reply score(john,ebft-a-subtraction-of-wholes,39);
Is has-taken-test(john,ebft-b-subtraction-of-wholes) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: no;

Trying the FAILURE version of explain.
Proved the FAILURE version of explain.

Trying the SUCCESS version of explain.
Proved the SUCCESS version of explain.

Task <subtraction-of-wholes> processed in Phase <analytic>
Success Theories: <[explain-fails]>
Bug Theories: <[problem(subtraction-of-wholes,john,espt-a)]>



Task = <sub-whole-1-digit-from-1-digit>


Trying the SUCCESS version of explain.
Is has-taken-test(john,est-b1) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: yes;
Is passed(john,est-b1) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: no;

Trying the FAILURE version of explain.
Proved the FAILURE version of explain.

Trying the SUCCESS version of explain.
Is not-passed(john,est-b1) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: yes;
Proved the SUCCESS version of explain.

Task <sub-whole-1-digit-from-1-digit> processed in Phase <analytic>
Success Theories: <[explain-fails]>
Bug Theories: <[problem(sub-whole-1-digit-from-1-digit,john,est-b1)]>

Prune at point <subtraction-of-numbers>



Task = <multiplication-of-numbers>


Task <multiplication-of-numbers> processed in Phase <analytic>
Success Theories: <[]>
Bug Theories: <[]>
```

```
Task = <multiplication-of-wholes>


Trying the SUCCESS version of explain.
Is has-taken-test(john,espt-a-multiplication-of-wholes) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: yes;
Is score(john,espt-a-multiplication-of-wholes,_36704) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: reply score(john,espt-a-multiplication-of-wholes,11);
Is has-taken-test(john,espt-b-multiplication-of-wholes) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: no;
Is has-taken-test(john,ebft-a-multiplication-of-wholes) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: yes;
Is score(john,ebft-a-multiplication-of-wholes,_37811) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: reply score(john,ebft-a-multiplication-of-wholes,38);
Is has-taken-test(john,ebft-b-multiplication-of-wholes) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: no;

Trying the FAILURE version of explain.
Proved the FAILURE version of explain.

Trying the SUCCESS version of explain.
Proved the SUCCESS version of explain.

Task <multiplication-of-wholes> processed in Phase <analytic>
Success Theories: <[explain-fails]>
Bug Theories: <[problem(multiplication-of-wholes,john,espt-a)]>

Prune at point <multiplication-of-numbers>


Task = <division-of-numbers>


Task <division-of-numbers> processed in Phase <analytic>
Success Theories: <[]>
Bug Theories: <[]>


Task = <division-of-wholes>


Trying the SUCCESS version of explain.
Is has-taken-test(john,espt-a-division-of-wholes) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: yes;
Is score(john,espt-a-division-of-wholes,_39222) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
```

```
Answer: reply score(john,espt-a-division-of-wholes,15);
Is has-taken-test(john,espt-b-division-of-wholes) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: no;
Is has-taken-test(john,ebft-a-division-of-wholes) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: yes;
Is score(john,ebft-a-division-of-wholes,_40674) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: reply score(john,ebft-a-division-of-wholes,41);
Is has-taken-test(john,ebft-b-division-of-wholes) true?
[Possible answers: yes; no; unknown; assume, reply <fact or rule>; help;]
Answer: no;

Trying the FAILURE version of explain.
Proved the FAILURE version of explain.

Trying the SUCCESS version of explain.
Proved the SUCCESS version of explain.

Task <division-of-wholes> processed in Phase <analytic>
Success Theories: <[explain-fails]>
Bug Theories: <[problem(division-of-wholes,john,espt-a)]>

Prune at point <division-of-numbers>


Prescriptive Phase

Prescriptive phase not implemented yet.


Diagnosis Concluded.

yes
?exit;
dragon[2]: Yes Master? exit
script done on Thu Jan 23 14:11:41 1986
```

APPENDIX B

TEST CASES

23. INTRODUCTION

Numerous unrecorded tests of the system occurred throughout the process of implementation. The five formal tests in this appendix are designed to demonstrate that the system actually works. The first two are non-clinical test cases that were designed to show that exploration of the task hierarchy, including pruning, worked properly. The last three are based upon clinical data. A diagnostician at the University of Saskatchewan, who worked closely with the subjects, tested them on the KeyMath and then extrapolated what their performance would be on the SDMT and Enright tests. Since WUP was not available at the University of Saskatchewan, where this testing was done, the results from these clinical cases had to be determined by tracing the program by hand.

The information for each test case consists of: the name of the phase of diagnosis; the name of the task being diagnosed; and the answers to give in response to the system's prompts. There may also be some text in italics describing why a certain action was taken during a test.

The first test case contains most of the explanations of the notation used, and the general significance of the test data. In the interest of brevity, the other test cases will not contain these explainations unless they illustrate some new aspect of the system.

24. CASE 1: JOHN

The John test case is not derived from clinical data, but is designed to test basic features of the program's operation, particularly the pruning rules.

RETROSPECTIVE Phase

Task math-problem:

name:	john
birth date:	1979 4 1
referral date:	1985 10 2
report date:	1985 11 13

Although, for generality, the system is designed to allow for queries for any task, in practice it was found that all the queries were at the very top node in the task hierarchy. Therefore, no replies are necessary for other tasks.

DEFINITIVE Phase

The system will ask many questions regarding the Enright tests. Most of the Enright tests have ''A'' and ''B'' forms. Answer ''yes;'' when the system asks if the student has taken the ''A'' version of the test, and ''no;'' if it asks about the ''B'' version. This saves time during the test and does not affect the results. If an answer for a test case does not indicate whether it is for the ''A'' for ''B'' form, assume that it is for the ''A'' form.

Task math-problem:
John is in grade 2.
Stanford Red (sdmt-red) score = 2.

''Stanford Red'' is short for the Stanford Diagnostic Mathematics Test, Red level. This is usually abbreviated ''sdmt-red''. The levels of the Stanford tests indicate the grade levels they are targetted at:

red for grades 1 to 4, and green for grades 4 to 6. (There is also a brown level for grades 6 to 12; it was not used for this system.) Many different scores are available from this test. The score sought for our purposes is the stanine score. Stanines range from 1 to 9; scores of 1 to 3 indicate below average performance; 4 to 6 indicates average performance; and 7 to 9 indicates above average performance. The Stanford tests also have a number of subtests; these are indicated accordingly.

```
Task operations:
        Stanford Red Computation Subtest: score = 2.

Task addition-of-numbers:
        NA.
```

Task nodes marked "NA" are Not Applicable during a diagnosis. For example, the addition-of-numbers *node exists only to serve as an abstract generalization for the nodes* addition-of-wholes, addition-of-fractions, *and* addition-of-decimals. *A task may also be NA if there are no parts of it that are applicable during a given phase. For example, nodes near the root of the hierarchy contain no information that is useful during the Analytic phase because the Analytic phase deals with detailed diagnoses rather than generalities.*

```
Task addition-of-wholes:
        Enright Wide Range Placement (EWRPT), form "A":
                error-in-ewrpt-a-questions-1-or-2 = yes.
                no-error-in-ewrpt-a-questions-1-or-2 = no.
        Keymath Addition Subtest:
                keymath-add-ceiling = 6.
```

The Enright Wide Range Placement test (ewrpt) has one page of questions about many different areas of elementary arithmetic. For addition, there are only two questions. An error in these questions indicates that a more comprehensive test of that particular skill should be administered. (Enright subtests are denoted by their initials, which always start with "e" (e.g., "ewrpt").) The KeyMath test is divided into a number of subtests. The information needed for the KeyMath is the number of the last question that the *administrator asked before ending the subtest.*

The information presented by this point should cause the search to be pruned at the point addition-of-numbers. Addition-of-numbers, *and all nodes beneath it in the task hierarchy, will no longer be searched during the Definitive phase. Pruning rules in an actual system would be more sophisticated than the ones given here.*

Task subtraction-of-numbers:
    NA.

Task subtraction-of-wholes:
    EWRPT:
        error-in-ewrpt-a-questions-3-or-4 = yes.
        no-error-in-ewrpt-a-questions-3-or-4 = no.
    KeyMath Subtraction Subtest:
        keymath-sub-ceiling = 7.

Task subtraction-of-integers:
    NA.

Task subtraction-of-fractions:
    EWRPT:
        error-in-ewrpt-a-questions-13-or-14 = yes.
        no-error-in-ewrpt-a-questions-13-or-14 = no.

Task subtraction-of-decimals:
    EWRPT:
        error-in-ewrpt-a-questions-21-or-22 = yes.
        no-error-in-ewrpt-a-questions-21-or-22 = no.

Task multiplication-of-numbers:
    NA.

Task multiplication-of-wholes:
    EWRPT:
        error-in-ewrpt-a-questions-5-or-6 = yes.
        no-error-in-ewrpt-a-questions-5-or-6 = no.
    KeyMath Multiplication Subtest:
        keymath-mult-ceiling = 3.

The search will be cut off at multiplication-of-numbers.

Task division-of-numbers:
    NA.

Task division-of-wholes:
    EWRPT:
        error-in-ewrpt-a-questions-7-or-8 = yes.
        no-error-in-ewrpt-a-questions-7-or-8 = no.
    KeyMath Division Subtest:
        keymath-div-ceiling = 2.

The search will be cut off at division-of-numbers.

ANALYTIC

The Analytic phase concentrates on more detailed aspects of the diagnosis; hence, no questions are asked of the user at the higher levels of the task hierarchy. Tasks at these levels are therefore denoted ''NA''.

Task math-problem:
    NA.

Task operations:
    NA.

Task addition-of-numbers:
    NA.

Task addition-of-wholes:
    ESPT:
        espt-a-addition-of-wholes score = 5.
    EBFT:
        ebft-a-addition-of-wholes score = 40.

The Enright Skill Placement Test and Basic Facts Test are more thorough tests that can give a diagnostician useful details about the child's arithmetic difficulties.

At this point, the search will be pruned at addition-of-numbers*; no further addition tasks will be examined.*

```
Task subtraction-of-numbers:
        NA.

Task subtraction-of-wholes:
        ESPT:
                espt-a-subtraction-of-wholes: score = 9.
        EBFT:
                ebft-a-subtraction-of-wholes: score = 39.

Task sub-whole-1-digit-from-1-digit:
        EST:
                not-passed(john,est-b1): yes.
                passed(john,est-b1): no.
```

The Enright Skill Test is the most detailed test in the Enright system, and it yields information useful in the Analytic phase of the diagnosis.

The search will be cut off at subtraction-of-numbers.

```
Task multiplication-of-numbers:
        NA.

Task multiplication-of-wholes:
        ESPT:
                espt-a-multiplication-of-wholes: score = 11.
        EBFT:
                ebft-a-multiplication-of-wholes: score = 38.
```

The search will be cut off at multiplication-of-numbers.

```
Task division-of-numbers:
        NA.

Task division-of-wholes:
        ESPT:
                espt-a-division-of-wholes: score = 15.
        EBFT:
                ebft-a-division-of-wholes: score = 41.
```

The search will be cut off at division-of-numbers.

25. CASE 2: JANELLE

The Janelle test case is not derived from clinical data, but is designed to test parts of the hierarchy that were pruned in Case 1. To do this, Janelle is said to have difficulty with subtraction of whole numbers when it involves the regrouping of both hundreds and tens.

There are fewer comments in the following cases; see Case 1 for details.

```
RETROSPECTIVE:

Task math-problem:
        name: janelle.
        birth date: 1978 11 5
        referral date: 1985 10 4
        report date: 1985 11 21

DEFINITIVE:

Task math-problem:
        Janelle is in Grade 3.
        Stanford Red (sdmt-red): score = 3.

Task operations:
        Stanford Red Computation Subtest: score = 3

Task addition-of-numbers:
        NA.

Task addition-of-wholes:
        EWRP:
                error-in-ewrpt-a-questions-1-or-2 = no.
                no-error-in-ewrpt-a-questions-1-or-2 = yes.
        KeyMath Addition Subtest:
                keymath-add-ceiling = 9.

Task addition-of-integers:
        NA.
```

Objects of type whole *are the non-negative integers; type* integer *was intended to be a superset of the wholes, including the negative integers. However, none of the tests that were used for the rules in this system had any components dealing with negative integers, so all the tasks dealing with integers are NA.*

Task addition-of-fractions:
  EWRPT:
    error-in-ewrpt-a-questions-11-or-12 = no.
    no-error-in-ewrpt-a-questions-11-or-12 = yes.

Task addition-of-decimals:
  EWRPT:
    error-in-ewrpt-a-questions-19-or-20 = no.
    no-error-in-ewrpt-a-questions-19-or-20 = yes.

Task subtraction-of-numbers:
  NA.

Task subtraction-of-wholes:
  EWRPT:
    error-in-ewrpt-a-questions-3-or-4 = yes.
    no-error-in-ewrpt-a-questions-3-or-4 = no.
  KeyMath Subtraction Subtest:
    keymath-sub-ceiling = 8.

Task subtraction-of-integers:
  NA.

Task subtraction-of-fractions:
  EWRPT:
    error-in-ewrpt-a-questions-13-or-14 = no.
    no-error-in-ewrpt-a-questions-13-or-14 = yes.

Task subtraction-of-decimals:
  EWRPT:
    error-in-ewrpt-a-questions-21-or-22 = no.
    no-error-in-ewrpt-a-questions-21-or-22 = yes.

Task multiplication-of-numbers:
  NA.

Task multiplication-of-wholes:
  EWRPT:
    error-in-ewrpt-a-questions-5-or-6 = no.
    no-error-in-ewrpt-a-questions-5-or-6 = yes.
  KeyMath Multiplication Subtest:
    keymath-mult-ceiling = 8.

Task multiplication-of-integers:
  NA.

Task multiplication-of-fractions:
  EWRPT:

error-in-ewrpt-a-questions-15-or-16 = no.
no-error-in-ewrpt-a-questions-15-or-16 = yes.

Task multiplication-of-decimals:
EWRPT:
error-in-ewrpt-a-questions-23-or-24 = no.
no-error-in-ewrpt-a-questions-23-or-24 = yes.

Task division-of-numbers:
NA.

Task division-of-wholes:
EWRPT:
error-in-ewrpt-a-questions-7-or-8 = no.
no-error-in-ewrpt-a-questions-7-or-8 = yes.
KeyMath Division Subtest:
keymath-div-ceiling = 10.

Task division-of-integers:
NA.

Task division-of-fractions:
EWRPT:
error-in-ewrpt-a-questions-17-or-18 = no.
no-error-in-ewrpt-a-questions-17-or-18 = yes.

Task division-of-decimals:
EWRPT:
error-in-ewrpt-a-questions-25-or-26 = no.
no-error-in-ewrpt-a-questions-25-or-26 = yes.

ANALYTIC

Task math-problem:
NA.

Task operations:
NA.

Task addition-of-number:
NA.

Task addition-of-wholes:
ESPT:
espt-a-addition-of-wholes: score = 11.
EBFT:
ebft-a-addition-of-wholes: score = 47.

Task addition-of-integers:
NA.

Task addition-of-fractions:
ESPT:
espt-a-addition-of-fractions: score = 9.

```
Task addition-of-decimals:
        ESPT:
                espt-a-addition-of-decimals: score = 10.

Task subtraction-of-numbers:
        NA.

Task subtraction-of-wholes:
        ESPT:
                espt-a-subtraction-of-wholes: score = 14.
        EBFT:
                ebft-a-subtraction-of-wholes: score = 48.

Task sub-whole-1-digit-from-1-digit:
        EST:
                not-passed(janelle,est-b1) = no.
                passed(janelle,est-b1) = yes.
```

The Enright Skill Tests are the most detailed tests in the Enright system. Each one covers a very specific skill; in the example given above, that skill is the subtraction of 1 digit whole numbers from 1 digit whole numbers.

```
Task sub-whole-1-digit-from-itself:
        EST:
                not-passed(janelle,est-b2) = no.
                passed(janelle,est-b2) = yes.

Task sub-whole-1-digit-from-2-digit:
        EST:
                not-passed(janelle,est-b4) = no.
                passed(janelle,est-b4) = yes.

Task sub-whole-1-digit-from-2-digit-no-regrouping:
        EST:
                not-passed(janelle,est-b5) = no.
                passed(janelle,est-b5) = yes.

Task sub-whole-1-digit-from-2-digit-regrouping-10s:
        EST:
                not-passed(janelle,est-b7) = no.
                passed(janelle,est-b7) = yes.

Task sub-whole-2-digit-from-2-digit:
        NA.

Task sub-whole-2-digit-from-2-digit-no-regrouping:
        EST:
                not-passed(janelle,est-b6) = no.
                passed(janelle,est-b6) = yes.

Task sub-whole-2-digit-from-2-digit-regrouping-10s:
        EST:
                not-passed(janelle,est-b8) = no.
                passed(janelle,est-b8) = yes.

Task sub-whole-3-digit-from-3-digit:
        NA.

Task sub-whole-3-digit-from-3-digit-no-regrouping:
        EST:
                not-passed(janelle,est-b9) = no.
                passed(janelle,est-b9) = yes.

Task sub-whole-3-digit-from-3-digit-regrouping-10s:
        EST:
                not-passed(janelle,est-b10) = no.
                passed(janelle,est-b10) = yes.

Task sub-whole-3-digit-from-3-digit-regrouping-100s:
        EST:
                not-passed(janelle,est-b11) = no.
                passed(janelle,est-b11) = yes.

Task sub-whole-3-digit-from-3-digit-regrouping-10s-&-100s:
        EST:
                not-passed(janelle,est-b12) = yes.
                passed(janelle,est-b12) = no.

Task subtraction-of-integers:
        NA.

Task subtraction-of-fractions:
        ESPT:
                espt-a-subtraction-of-fractions: score = 8.

Task subtraction-of-decimals:
        ESPT:
                espt-a-subtraction-of-decimals: score = 10.

Task multiplication-of-numbers:
        NA.

Task multiplication-of-wholes:
        ESPT:
                espt-a-multiplication-of-wholes: score = 16.
        EBFT:
                ebft-a-multiplication-of-wholes: score = 47.

Task multiplication-of-fractions:
        ESPT:
                espt-a-multiplication-of-fractions: score = 6.

Task multiplication-of-decimals:
        ESPT:
                espt-a-multiplication-of-decimals: score = 5.
```

Task division-of-numbers:
NA.

Task division-of-wholes:
ESPT:
espt-a-division-of-wholes: score = 21.
EBFT:
ebft-a-division-of-wholes: score = 49.

Task division-of-integers:
NA.

Task division-of-fractions:
ESPT:
espt-a-division-of-fractions: score = 6.

Task division-of-decimals:
ESPT:
espt-a-division-of-decimals: score = 9.

26. CASE 3: ASHLEY

Ashley is the first of three cases that are derived from clinical data. In each case, the subject had only received the KeyMath test, but an experienced diagnositician from the University of Saskatchewan, who had worked with the subjects, extrapolated his or her performance to Enright Wide Range Placement Test and the Enright Skill Placement Test.

RETROSPECTIVE:

Task math-problem:
name: ashley
birth date: 1976 11 11
referral date: 1985 12 15
report date: 1985 3 31

DEFINITIVE:

Task math-problem:
Ashley is in Grade 5.
Stanford Green (sdmt-green): score = 3.

Task operations:
Stanford Green Computation Subtest: score = 2.

Task addition-of-numbers:
NA.

Task addition-of-wholes:

EWRP:
error-in-ewrpt-a-questions-1-or-2 = yes.
no-error-in-ewrpt-a-questions-1-or-2 = no.
KeyMath Addition Subtest:
keymath-add-ceiling = 6.

search will be pruned at addition-of-numbers.

subtraction-of-numbers:
NA.

subtraction-of-wholes:
EWRPT:
error-in-ewrpt-a-questions-3-or-4 = yes.
no-error-in-ewrpt-a-questions-3-or-4 = no.
KeyMath Subtraction Subtest:
keymath-sub-ceiling = 7.

search is cut at subtraction-of-numbers.

k multiplication-of-numbers:
NA.

k multiplication-of-wholes:
EWRPT:
error-in-ewrpt-a-questions-5-or-6 = yes.
no-error-in-ewrpt-a-questions-5-or-6 = no.
KeyMath Multiplication Subtest:
keymath-mult-ceiling = 0.

ch is pruned at multiplication-of-numbers.

division-of-numbers:
NA.

division-of-wholes:
EWRPT:
error-in-ewrpt-a-questions-7-or-8 = yes.
no-error-in-ewrpt-a-questions-7-or-8 = no.
KeyMath Division Subtest:
keymath-div-ceiling = 2.

LYTIC

math-problem:
NA.

operations:
NA.

addition-of-number:
NA.

addition-of-wholes:

ESPT:
espt-a-addition-of-wholes: score = 4.

Search is pruned at addition-of-numbers.

Task subtraction-of-wholes:
ESPT:
espt-a-subtraction-of-wholes: score = 6.

Since performance was not extrapolated to the Enright Skills Test, there is no mark available for any of its subtests. All the theories for the specializations of subtraction-of-wholes *depend upon these subtests. When the system asks if the student has taken such a subtest, answer* no. *Since the same response is expected for each specialization of* subtraction-of-wholes, *they are omitted from here.*

Task subtraction-of-integers:
NA.

Task subtraction-of-fractions:
ESPT:
espt-a-subtraction-of-fractions: score = 0.

earch is pruned at subtraction-of-fractions.

Task subtraction-of-decimals:
ESPT:
espt-a-subtraction-of-decimals: score = 0.

earch is pruned at subtraction-of-decimals.

ask multiplication-of-numbers:
NA.

ask multiplication-of-wholes:
ESPT:
espt-a-multiplication-of-wholes: score = 0.

earch is pruned at multiplication-of-numbers.

ask division-of-numbers:
NA.

ask division-of-wholes:
ESPT:
espt-a-division-of-wholes: score = 0.

arch is pruned at division-of-numbers.

27. *CASE 4: SUE*

RETROSPECTIVE:

Task math-problem:
name: sue
birth date: 1974 6 18
referral date: 1985 10 1
report date: 1985 12 2

DEFINITIVE:

Task math-problem:
Sue is in Grade 7. (Use Grade 6 to get the system to work.)
Stanford Green (sdmt-green): score = 3.

Task operations:
Stanford Green Computation Subtest: score = 3.

Task addition-of-numbers:
NA.

Task addition-of-wholes:
EWRP:
error-in-ewrpt-a-questions-1-or-2 = no.
no-error-in-ewrpt-a-questions-1-or-2 = yes.
KeyMath Addition Subtest:
keymath-add-ceiling = 12.

Search is pruned at addition-of-wholes.

Task addition-of-integers:
NA.

Task addition-of-fractions:
EWRPT:
error-in-ewrpt-a-questions-11-or-12 = yes.
no-error-in-ewrpt-a-questions-11-or-12 = no.

Search is cut at addition-of-fractions.

Task addition-of-decimals:
EWRPT:
error-in-ewrpt-a-questions-19-or-20 = no.
no-error-in-ewrpt-a-questions-19-or-20 = yes.

Search is cut at addition-of-decimals.

Task subtraction-of-numbers:
NA.

Task subtraction-of-wholes:
EWRPT:

error-in-ewrpt-a-questions-3-or-4 = yes.
no-error-in-ewrpt-a-questions-3-or-4 = no.
KeyMath Subtraction Subtest:
keymath-sub-ceiling = 9.

Task subtraction-of-integers:
NA.

Task subtraction-of-fractions:
EWRPT:
error-in-ewrpt-a-questions-13-or-14 = yes.
no-error-in-ewrpt-a-questions-13-or-14 = no.

Search is pruned at subtraction-of-fractions.

Task subtraction-of-decimals:
EWRPT:
error-in-ewrpt-a-questions-21-or-22 = yes.
no-error-in-ewrpt-a-questions-21-or-22 = no.

Task multiplication-of-numbers:
NA.

Task multiplication-of-wholes:
EWRPT:
error-in-ewrpt-a-questions-5-or-6 = no.
no-error-in-ewrpt-a-questions-5-or-6 = yes.
KeyMath Multiplication Subtest:
keymath-mult-ceiling = 8.

Search is pruned at multiplication-of-wholes.

Task multiplication-of-integers:
NA.

Task multiplication-of-fractions:
EWRPT:
error-in-ewrpt-a-questions-15-or-16 = yes.
no-error-in-ewrpt-a-questions-15-or-16 = no.

Search is pruned at multiplication-of-fractions.

Task multiplication-of-decimals:
EWRPT:
error-in-ewrpt-a-questions-23-or-24 = yes.
no-error-in-ewrpt-a-questions-23-or-24 = no.

Task division-of-numbers:
NA.

Task division-of-wholes:
EWRPT:
error-in-ewrpt-a-questions-7-or-8 = yes.
no-error-in-ewrpt-a-questions-7-or-8 = no.
KeyMath Division Subtest:

```
                keymath-div-ceiling = 3.

Search is pruned at division-of-numbers.


ANALYTIC

Task math-problem:
        NA.

Task operations:
        NA.

Task addition-of-number:
        NA.

Task addition-of-wholes:
        ESPT:
                espt-a-addition-of-wholes: score = 10.

Search is pruned at addition-of-wholes.

Task addition-of-integers:
        NA.

Task addition-of-fractions:
        ESPT:
                espt-a-addition-of-fractions: score = 5.

Search is pruned at addition-of-fractions.

Task addition-of-decimals:
        ESPT:
                espt-a-addition-of-decimals: score = 11.

Search is pruned at addition-of-decimals.

Task subtraction-of-numbers:
        NA.

Task subtraction-of-wholes:
        ESPT:
                espt-a-subtraction-of-wholes: score = 11.

Enright Skill Test scores are not available.

Task subtraction-of-integers:
        NA.

Task subtraction-of-fractions:
        ESPT:
                espt-a-subtraction-of-fractions: score = 2.
```

```
'arch pruned at subtraction-of-fractions.

ısk subtraction-of-decimals:
      ESPT:
              espt-a-subtraction-of-decimals: score = 5.

ısk multiplication-of-numbers:
      NA.

ısk multiplication-of-wholes:
      ESPT:
              espt-a-multiplication-of-wholes: score = 15

arch pruned at multiplication-of-wholes.

ısk multiplication-of-fractions:
      ESPT:
              espt-a-multiplication-of-fractions: score = 2.

arch pruned at multiplication-of-fractions.

ısk multiplication-of-decimals:
      ESPT:
              espt-a-multiplication-of-decimals: score = 2

ısk division-of-numbers:
      NA.

sk division-of-wholes:
      ESPT:
              espt-a-division-of-wholes: score = 1.

ırch pruned at division-of-numbers.
```

. CASE 5: CAM

```
TROSPECTIVE:

ık math-problem:
      name: cam
      birth date: 1974 3 15
      referral date: 1985 9 15
      report date: 1985 12 15

FINITIVE:
```

Task math-problem:
    Cam is in Grade 7. (Use Grade 6 to get the system to work.)
    Stanford Green (sdmt-green): score = 1.

Task operations:
    Stanford Green Computation Subtest: score = 1.

Task addition-of-numbers:
    NA.

Task addition-of-wholes:
    EWRP:
        error-in-ewrpt-a-questions-1-or-2 = yes.
        no-error-in-ewrpt-a-questions-1-or-2 = no.
    KeyMath Addition Subtest:
        keymath-add-ceiling = 9.

Search is pruned at addition-of-numbers.

Task subtraction-of-numbers:
    NA.

Task subtraction-of-wholes:
    EWRPT:
        error-in-ewrpt-a-questions-3-or-4 = yes.
        no-error-in-ewrpt-a-questions-3-or-4 = no.
    KeyMath Subtraction Subtest:
        keymath-sub-ceiling = 5.

Task subtraction-of-integers:
    NA.

Task subtraction-of-fractions:
    EWRPT:
        error-in-ewrpt-a-questions-13-or-14 = yes.
        no-error-in-ewrpt-a-questions-13-or-14 = no.

Search is pruned at subtraction-of-fractions.

Task subtraction-of-decimals:
    EWRPT:
        error-in-ewrpt-a-questions-21-or-22 = yes.
        no-error-in-ewrpt-a-questions-21-or-22 = no.

Task multiplication-of-numbers:
    NA.

Task multiplication-of-wholes:
    EWRPT:
        error-in-ewrpt-a-questions-5-or-6 = yes.
        no-error-in-ewrpt-a-questions-5-or-6 = no.
    KeyMath Multiplication Subtest:
        keymath-mult-ceiling = 5.

Search is pruned at multiplication-of-numbers.

Task division-of-numbers:
    NA.

Task division-of-wholes:
    EWRPT:
        error-in-ewrpt-a-questions-7-or-8 = yes.
        no-error-in-ewrpt-a-questions-7-or-8 = no.
    KeyMath Division Subtest:
        keymath-div-ceiling = 3.

Search is pruned at division-of-numbers.

ANALYTIC

Task math-problem:
    NA.

Task operations:
    NA.

Task addition-of-number:
    NA.

Task addition-of-wholes:
    ESPT:
        espt-a-addition-of-wholes: score = 9.

Search is pruned at addition-of-numbers.

Task subtraction-of-numbers:
    NA.

Task subtraction-of-wholes:
    ESPT:
        espt-a-subtraction-of-wholes: score = 6.

Enright Skill Test scores are not available.

Task subtraction-of-integers:
    NA.

Task subtraction-of-fractions:
    ESPT:
        espt-a-subtraction-of-fractions: score = 0.

```
Search pruned at subtraction-of-fractions.

Task subtraction-of-decimals:
        ESPT:
                espt-a-subtraction-of-decimals: score = 1.

Task multiplication-of-numbers:
        NA.

Task multiplication-of-wholes:
        ESPT:
                espt-a-multiplication-of-wholes: score = 1.

Search pruned at multiplication-of-numbers.

Task division-of-numbers:
        NA.

Task division-of-wholes:
        ESPT:
                espt-a-division-of-wholes: score = 1.

Search pruned at division-of-numbers.
```