Intelligent Algebraic Tutoring Based on Student Misconceptions

by

Brian Grossman

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degrees of Bachelor of Science in Computer Science and Engineering and Master of Engineering in Electrical Engineering and Computer Science at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 7, 1996

Author .....
Department of Electrical Engineering and Computer Science
February 7. 1996

Certified by .....
John V. Guttag
Thesis Supervisor

Accepted by .....
F. R. Morgenthaler
Chairman, Department Committee on Graduate Theses

Intelligent Algebraic Tutoring Based on Student Misconceptions

by

Brian Grossman

Submitted to the Department of Electrical Engineering and Computer Science on February 7, 1996, in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering and Master of Engineering in Electrical Engineering and Computer Science.

Abstract

The purpose of this project was to develop algebraic tutoring software that recognizes common student mistakes and uses that information to supply helpful hints. Teachers can use this software to help teach their students learn to solve first-order, single-variable equations with integer solutions. They supply the program with a list of common student misconceptions described in an error language. The program then compares these descriptions to students' input to determine if particular mistakes have been made. If so, helpful hints, also input by the teachers, are given to the students. The program also keeps a log of the students' sessions that may be reviewed by teachers.

Thesis Supervisor: John V. Guttag
Title: Professor

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recently the educational software market has flourished. Many groups are now researching and developing the use of computers as teaching tools in both schools and homes. This technology has some distinct advantages over standard teaching.

Unlike a human teacher, a student can learn from his computer in a one-on-one setting at any time of the day or night. Any teacher or student who has been in an overcrowded classroom environment would surely appreciate this convenience. A student is able to learn at a computer as an individual, at his own pace. As long as the student is interested in learning the subject matter, educational software can be a useful tool in picking up education where a teacher may be forced to leave off because of time constraints. Educational software can also be more effective than textbooks because it can provide a higher level of interactivity. Students can supply input to a computer and receive feedback.

To best utilize these advantages, it is important that educational software be highly interactive. Programs should be able to interpret a student's correct answers as well as incorrect responses. They should also supply intelligent feedback to the students, and most of all, they should be able to effectively convey concepts to their users.

The purpose of this project was to research new techniques for computer-based algebraic tutoring. One of the two main goals of this research was to develop a language in which common student errors can be described so that they may be diagnosed. The other was to involve teachers in the computer learning process. The first goal was made possible by implementing an input language that enables the Tutor to calculate what students' inputs would be if they made particular mistakes. These calculations are then compared to

actual student inputs to recognize their errors. The second goal was achieved by designing the Tutor to run problems compiled by teachers and to allow teachers to edit files that contain information necessary to diagnose common errors. Teachers are also able to check on the progress of students by reviewing log files created by the Tutor.

The remainder of this chapter first describes two examples of related projects and then how this project attempts to realize its goals of developing an error language and involving teachers in the learning process. Chapter 2 describes how the Tutor was designed. It explains how both students and teachers interact with the Tutor. Chapter 3 discusses the results of testing the Tutor and its problem areas. A summary of the project, along with ideas for further research and a conclusion are presented in Chapter 4.

1.1 Common Misconceptions

Teaching new concepts to people can be extremely difficult. Unlike general knowledge, mathematical concepts cannot be conveyed to students with words alone. Successful teaching of a concept requires the learner to not only understand the concept but also to be able to apply it in varying situations. To do so, the learner must establish an abstract understanding of the notion in his mind, as opposed to simply storing data, as with general knowledge.

When a student starts learning a mathematical concept, he tends to make more errors than someone who has had more experience with the material. Fortunately, these mistakes tend to be somewhat predictable using knowledge that the student already has. This is due to the fact that, when faced with unfamiliar kinds of math problems, students will generally try to make intelligent guesses based on prior experience before making arbitrary assumptions.

Much research has been done on the subject of creating techniques for teaching mathematical concepts by utilizing common misconceptions. M. Matz, of the M.I.T. Artificial Intelligence Laboratory did research on exactly this subject. In her paper entitled, "Towards a Process Model for High School Algebra Errors" [6], she attempts to account for the uniformity of the errors students make in solving algebra problems. Another project, named "Diagnostic Models for Procedural Bugs in Basic Mathematical Skills" [1] attempts to explain students' addition and subtraction errors.

1.1.1 Towards a process model for high school algebra

In this paper, Matz breaks down common errors into three main categories:

- 1. Errors generated by an incorrect choice of an extrapolation technique
- 2. Errors reflecting the conceptual changes needed to learn algebra after knowing arithmetic
- 3. Errors arising during the execution of a procedure.

1.1.1.1 Extrapolation

Matz defines extrapolation as the technique a student uses when trying to solve a new problem without an applicable rule. The student attempts to apply a familiar but not necessarily applicable rule to the problem. This frequently results in a describable, diagnoseable mistake. An example of this would be the following:

- Simplify: A(B*C)
- Student knows: A(B+C)=AB+AC
- Student guesses: A(B*C)=AB*AC

Since the student knows that the A can be distributed over a parenthetical expression which is the sum of two numbers, he assumes that the A can be distributed over a parenthetical expression which is the product of two numbers. This is obviously not true. (Consider A=2, B=3, C=4. A(B*C) is 24, while AB*AC is 48) The student tries to solve an unfamiliar problem by applying a similar, more familiar rule. This mistake is not arbitrary and can easily be recognized.

1.1.1.2 Conceptual Changes

For a student who only knows arithmetic, learning algebra not only requires learning the rules of algebra, but also its underlying concepts. The introduction of the variable is one of these concepts. Students must be able to understand that the 'x' in the equation x=3+4 is a place holder for an actual value. It represents a value equal to 7. Similar problems arise with the introduction of algebraic notation and equality. These are initially very awkward for the student and, like extrapolations, tend to result in common misconceptions.

One example of the type of mistake that would result from conceptual changes would be the following:

- Solve: 3x=36
- Student guesses: x=6

In this case, the student guesses that x=6 because the concatenation of "3" and "6" is "36". The student did not realize that 3x is the notation for 3*x where x is a place holder for a value. (In this case, x=12 because 3*12=36) Like extrapolation errors, conceptual change errors can also be recognized.

1.1.1.3 Procedure Execution

Matz defines procedure execution errors as errors in planning or processing. Planning errors are errors in which the student has difficulty executing the correct procedures in order to achieve the desired result, for example, "solve" or "simplify". To avoid planning errors, the student must know the goal that he is trying to reach. The student must know that "solving" a problem means finding the value for the variable in the equation. For example, if the student was asked to solve 4x-2=10, the answer 4x=12 would not be complete. The student has not made any mistakes, but he has misinterpreted the goal of "solving" an equation. The student has not completed the problem. If the student suffers from planning errors, he may not be making any mistakes; it is possible that he is just unable to get the right answer without understanding the goals of the problem.

Processing errors, unlike all others types of errors, are considered to be errors of execution rather than conceptual errors. These are errors that "reflect slips of performance rather than real misunderstandings." Matz suggests that even though these mistakes are solely due to the student's processing errors, they too can be described and diagnosed. She also suggests that many of these errors occur when a student attempts to do more than one step at a time. In the student's haste, he incorrectly executes a substep in his head and generates an incorrect guess. Many of these mistakes are also fairly common and therefore recognizable.

1.1.2 Diagnostic Models for Procedural Bugs in Basic Mathematical Skills

In "Diagnostic Models for Procedural Bugs in Basic Mathematical Skills", John Seely Brown and Richard R. Burton attempt to establish "a mechanism for explaining why a student is making a mistake as opposed to simply identifying the mistake." This approach to finding mistakes can prove to be invaluable for teaching. If a tutor (human or computer) is not only aware of the mistakes made by a student, but also aware of why they are made, he (it) has a better chance of giving the student a useful suggestion to avoid such errors in the future. As a result, the student can learn why what he did was a mistake, and how to avoid such errors. This intelligent feedback is extremely effective in curbing student misconceptions before they become habit.

Unlike Matz, who generalized a broad range of algebraic misconceptions, Brown and Burton attempted to enumerate a set of specific errors in addition and subtraction. They narrowed their scope so that they could more easily implement a program that would recognize mistakes. They suggested that attempting to implement a program that can diagnose errors in algebra would prove to be much more difficult. They first compiled some data by drilling students with addition and subtraction problems. They then analyzed the information and attempted to generalize mistakes that the students made. After doing so, they quizzed the students again, this time with problems that would be done incorrectly if they were to make the same mistakes they did in the first test.

Their data revealed, much to the surprise of many educators, that "students are remarkably competent procedure followers, but they often follow the wrong procedures." This type of mistake could be a result of what Matz classifies as an extrapolation error. Brown and Burton then set out to construct a "deep-structure model of the student's errors

in a set of buggy subprocedures that, when invoked, replicates those errors." This model contained the information necessary to reproduce the students' common misconceptions.

Brown and Burton then created a program to help people learn to diagnose these common arithmetic bugs. The program would start by giving an example of a problem and an incorrect solution. This incorrect guess at a solution was calculated by the program and exhibited a particular bug. The users then input new examples for the program to solve. The program then made the same mistake in solving the new examples as it did in solving the original example. The users were then asked to notify the computer when they thought that they understood the bug the program was making. To check that the users had successfully diagnosed the bug, the computer would give them examples to try to solve incorrectly by applying the bug. If this was done correctly, the computer then gave a text description of the misconception. This technique proved very effective for both teachers and students alike in diagnosing common misconceptions in arithmetic.

Brown and Burton suggest that this program, BUGGY, could be used to help train teachers to understand and diagnose these common misconceptions. Like Matz, Brown and Burton suggest that most of the mistakes made by students in mathematics are not random, but describable, diagnoseable misconceptions.

1.1.3 Application of Prior Research

This project, Intelligent Algebraic Tutoring Based on Student Misconceptions, attempts to utilize much of the work done by Matz, Brown, Burton, and others to provide the student with an intelligent algebraic tutor. The entirety of this project assumes that student misconceptions exist for a reason and are therefore describable and diagnoseable. The Tutor provides teachers with a flexible language with which they can describe com-

mon algebraic misconceptions of students. Along with describing algebraic errors, the language also allows teachers to map helpful hints to the set of common errors. During the course of operation, the program checks the student's input against the list of common mistakes. If the program determines that one or more of the common errors has been made by the student, it yields the corresponding hint(s).

To be an effective tool for recognizing common mistakes in solving first-order, single-variable algebraic equations, the Tutor needed to apply features from both Matz's work and BUGGY. An ideal Tutor would be able to recognize all the different types of misconceptions presented in Matz's paper. The Tutor was therefore designed to be able to predict many types of errors, regardless of classification. This was done by checking if the student's result matched the result that would be obtained if a particular mistake was made. In this way, many types of mistakes could be applied to a particular problem to determine if the student made one or more of them. In this way, the Tutor was similar to BUGGY. The Tutor was able to apply many mistakes to specific examples and determine the results of their application.

1.2 Step-by-Step Problem Solving

Solving algebraic equations is not a single-step process. To solve first-order, single-variable equations, for example, an individual must first learn approximately seven new, abstract, algebraic rules. He must then be able to sequentially apply the appropriate ones until he has simplified the equation to the form x=c (where x is the variable and c is a constant). There is not one specific, correct way to solve these equations. The abstract rules can be applied in various sequences to obtain the correct answer. [See Figure 1.1] This requires both understanding of algebraic concepts and knowledge of when to apply them.

It was therefore important to create an environment in which the student is able to step through the problems on the computer.

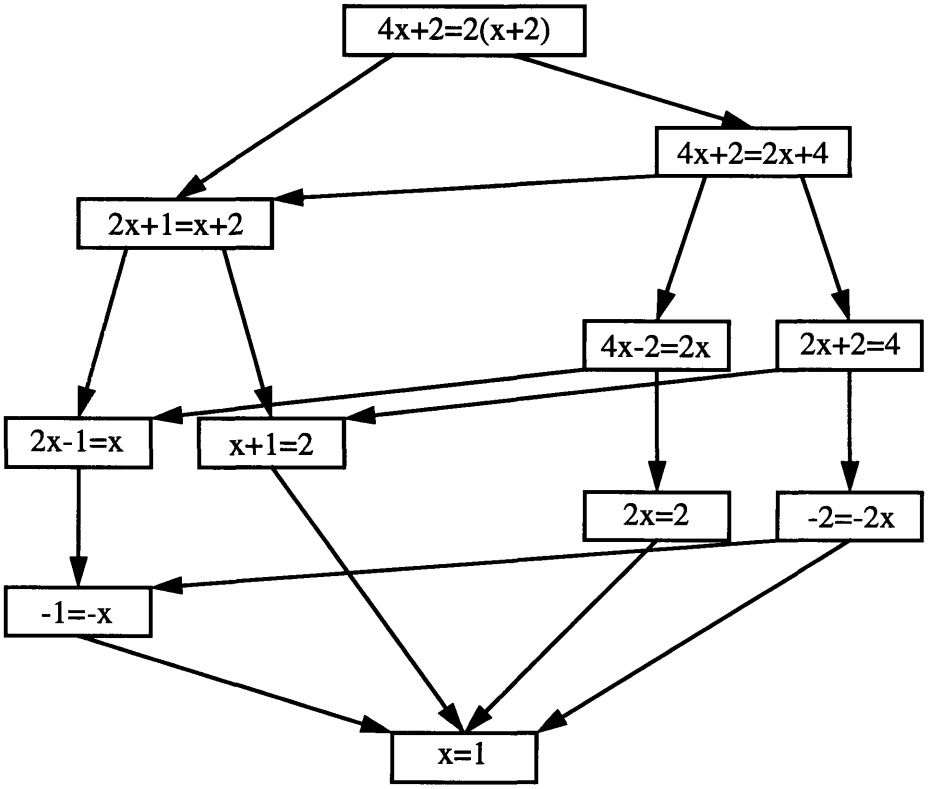


Figure 1.1: Possible Methods of Solving 4x+2=2(x+2)

The Tutor demonstrates flexibility when accepting the student's guesses. After he is given a new problem, he is then asked to either enter the solution or to enter a step toward the solution. This technique gives the student freedom and allows him to work out entire problems step-by-step with the Tutor instead of using scratch paper.

1.2.1 Enter-The-Answer

Some mathematical tutoring software uses the enter-the-answer approach to problem solving. In this situation, the student is given a problem and asked to work it out by hand

and/or in his head. He is then instructed to input the result. The program then determines if the result is correct. If it is, the program congratulates the student, otherwise, the program will generally show the student one possible way to step through the problem to obtain the correct answer. This would not have been beneficial for the Tutor for two main reasons.

Firstly, with the enter-the-answer approach, students are exposed to only one possible solution to a given problem. Anyone who has learned algebra knows that with any given problem, there are numerous correct paths to get to the correct answer. A program should not limit a student's ability to explore different techniques for problem solving. Students should be allowed to experiment with various methods. This encouragement of creativity and exploration early on in the teaching process will inevitably lead to a deeper understanding of the concepts that lie behind the mechanical algorithms of algebra. Individuals who explore different problem solving techniques will be better able to solve unfamiliar problems. The knowledge of an abstract mathematical concept greatly outweighs the knowledge of the mechanical operations that reflect it.

The second reason why the step-by-step method is superior is because it makes diagnosing misconceptions possible. It is nearly impossible to create an algebraic tutor that is able to diagnose a student's error if all of the work is done on scratch paper and/or in his head and only the answer is entered. By allowing students to solve problems step-by-step on the computer screen, the Tutor has a window into the student's thought process. The Tutor is able to pinpoint the exact step in which the student is making his mistake. The Tutor then is able to diagnose the student's misconception with that particular step and give an intelligent hint.

This diagnosis of a particular step also has another advantage. The Tutor is able to make the student aware of his mistake as soon as he takes an incorrect step. From a psychological standpoint, this is very beneficial. Since the student is made aware of his mis-

take immediately, he is less likely to repeat it, preventing the student from forming bad habits in problem solving. The enter-the-answer technique would not accomplish this. The student could make an incorrect step on scratch paper, enter his guess at the solution, and be told only that he has made some mistake somewhere in his thought process.

1.2.2 Multiple Choice

Another common approach to problem solving commonly found in software is multiple choice answers. In this approach, the equation is given, along with several possible answers. The student is then asked to solve the problem on scratch paper and/or in his head and choose the answer that matches his solution. This approach has many of the same drawbacks as the enter-the-answer approach as well as some additional ones.

The multiple choice approach does have the advantage of interpreting some possible misconceptions. For example, if of four possible choices, one was the correct answer and the other three were answers that the student would arrive at if he had a particular misconception, the program could diagnose a limited set of problems. This is effective to a point. However, if the student makes a mistake that is not reflected in one of the choices, he will probably be unable to select one of the choices. The student will know that he has done something wrong, but will have no idea how to fix it.

1.3 Teacher Involvement

Another advantage of the Tutor is that it directly involves the teacher in the learning process, utilizing the Tutor utilizes the knowledge of those who are best qualified to teach mathematics--the teachers. Teachers are involved in three aspects of the program: the examples, the errors and the log files.

1.3.1 Examples

With the Tutor, a teacher can easily create an entire assignment for his students. To create an assignment of examples to be run by the Tutor, a teacher would need only to enter them in a separate text file. [See Section 2.2.1] This file would be loaded by the program and examples would be given to the student, one at a time. In this way, the teacher can integrate the Tutor into his established curriculum by simply creating assignments with equations at the appropriate difficulty level and complexity for his students. Along with the equations, the assignment files also have the ability to display comments to the student.

1.3.2 Errors

Along with creating assignment files, the teacher is allowed to edit files that contain common misconceptions. [See Section 2.2.2] He can add new errors to the error files as necessary. The teacher can also reword existing hints. In this way, the Tutor can be perceived of as an extension of the teacher. The Tutor can be programmed to detect many mistakes that the teacher would look for and would also give hints in the teacher's own words. This advantage eliminates the need for the teacher to monitor every step each student takes. A teacher can assign homework on the Tutor and be assured that students will receive coaching in familiar words.

The other advantage of teacher involvement in the creation of the error descriptions is that it forces the teacher to think like students. In order to enter a new error description to the error file, the teacher must be able to understand why a student would make a particu-

lar error. This is very important because a teacher who can learn to predict the mistakes his students are likely to make will become a better teacher. He can not only teach students the right way to solve problems, but can also show them why common misconceptions are incorrect--and the earlier in the learning process that the students learn to avoid common mistakes, the better.

1.3.3 Log Files

The final method in which the Tutor involves the teacher is with log files. Every time a student uses the Tutor, it keeps a complete log of student entries and Tutor outputs. The entire session is stored in a file. [See Section 2.2.3] The teacher can easily review these logs to determine where students are having problems and track their progress. He can assign homework in the form of a problem file and collect it in the form of a log file. He can then carefully analyze the students' results and add appropriate errors to the error file.

Chapter 2

Design

The Tutor's design can be broken down into two subsystems, the Student Subsystem and the Teacher Subsystem. The Student Subsystem interacts directly with the student. It displays necessary information for the student, and also reads student input as described in the Student Subsystem section. The Teacher Subsystem interacts with the teacher. With the Tutor, teachers are able to create examples for students to solve, generate hints for students when make particular mistakes are made, and review the work of their students. These features are described in the Teacher Subsystem section. The final section of this chapter will describe the design issues involved in evaluating the steps taken by students.

2.1 Student Subsystem

When a student starts a session with the Tutor, he is initially prompted to enter his name, the name of the file that contains his assignment, the name of the file that contains hints for common mistakes, and the name of the log file. Then the Tutor loads the first example from the assignment file and prints it to the screen for the student. The flow of a student's session with the Tutor is summarized in Figure 2.1.

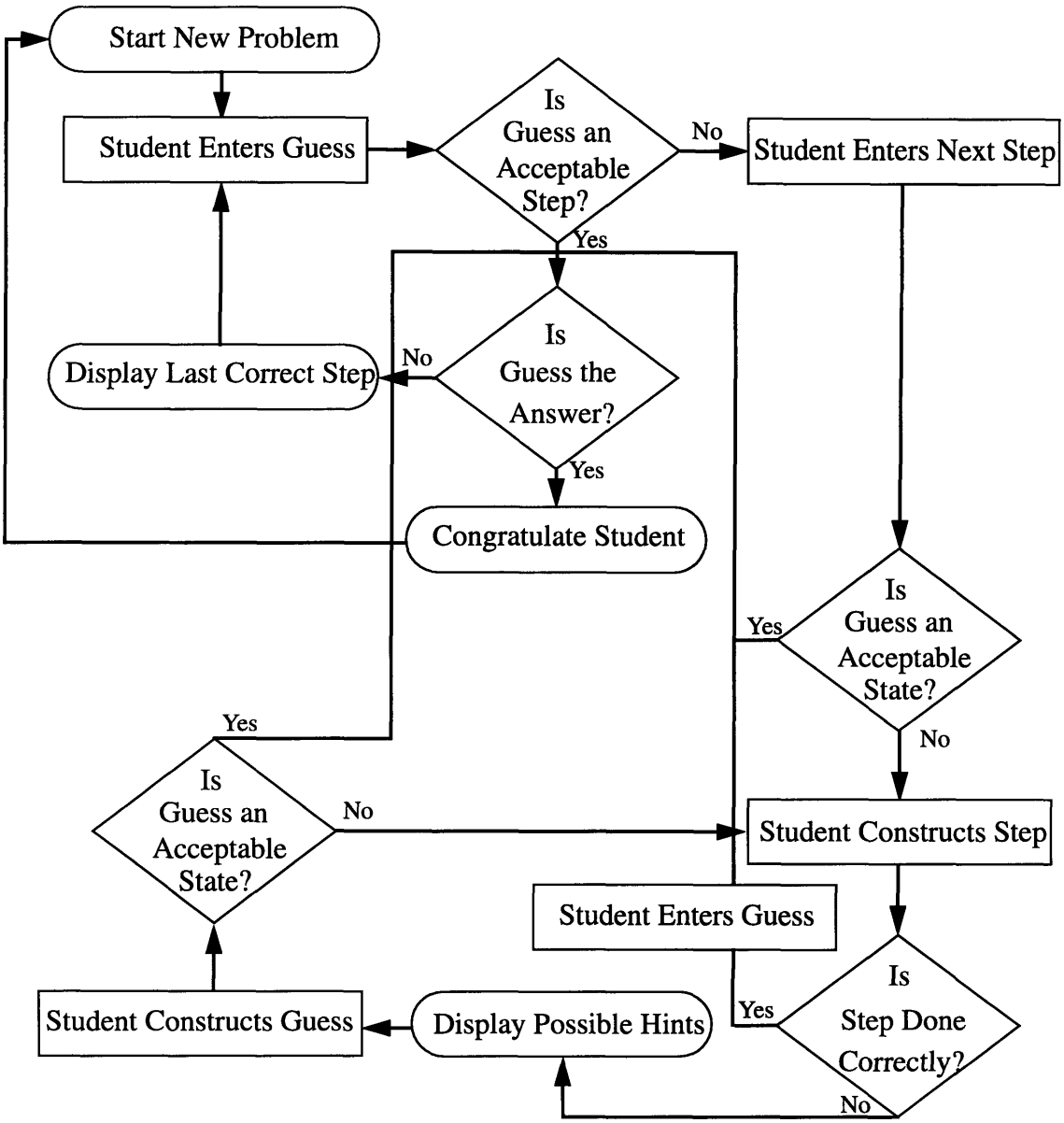


Figure 2.1: Student Flow Chart

After reading the first problem, the student may either enter the solution to the equation (of the form x=c) or enter a step towards this solution. The Tutor then checks to deter-

mine if the student's input is acceptable. If it is the correct answer, the student is congratulated and he may begin the next example. [See Figure 2.2]

```
The equation is: 1 + 1 = 2x

Enter a guess: (Hit return to quit)
x=1
Very Good, You Solved the Problem Correctly

Would you like to do another example? (Y or N)
y
```

Figure 2.2: Solving a Problem

If the student's input is not the answer but is an acceptable step towards the solution, the Tutor informs the student that he is on the right track, reprints the equation, reprints the last correct step, and prompts the student for his next guess. [See Figure 2.3] This technique allows students to solve equations step-by-step without scratch paper. [See Section 1.2.1] A student is able to do any set of operations to the equation, provided they are legal by the rules of algebra. For example, if the student arbitrarily decides to add 100 to both sides of the equation, the Tutor will accept this step. In this way, the Tutor accepts a set of guesses that may contain equations that, although equivalent in answer, are not efficient steps to take in problem solving. [See Section 2.3]

```
The equation is: 3 * 2x + 6 = 24

Enter a guess: (Hit return to quit)
6x+6=24
Good, You're on the right track, keep going
```

Figure 2.3: Taking a Step

If the student's guess is not acceptable, the Tutor informs him that he has made a mistake. It then asks the student to try again, taking only one step at a time. If the student recognizes his mistake, he lets the Tutor know this by entering "Y". [See Figure 2.4] The Tutor then ignores the mistake and prompts the student for a new guess as if no mistake had been made. This feature is helpful if the student suddenly understands his error or makes a typing mistake. Often, just knowing that he has made a mistake will allow the student to determine what he has done wrong.

```
The equation is: 3 * 2x + 6 = 24
Your last correct step was: 6x + 6 = 24

Enter a guess: (Hit return to quit)
2x+6=8

You have made a mistake.
Your last correct step was: 6x + 6 = 24
Try again, but make sure you only take one step at a time.
If you understand your error at any time, type "Y"
```

Figure 2.4: Making a Mistake

If the student does not understand his error, he now needs to solve the problem one step at a time. There are seven steps that are recognized by the Tutor. [See Figure 2.5]

- combining two terms by adding
- combining two terms by subtracting
- combining two terms by multiplying
- combining two terms by distributing over parenthesis
- moving a term to the other side of the equation
- dividing the equation by a term
- removing parenthesis from a parenthetical expression

Figure 2.5: Possible Steps

If the student's next step is correct, the Tutor once again prompts him to take another single step. This continues until the student has made a mistake or solved the problem. In this way, the Tutor can determine the single step in which the student is erring. The Tutor then lists the above mentioned possible steps and asks the student to choose the one which he is attempting. [See Figure 2.6]

```
2x+6=8

You are making your mistake on this step.

The equation is: 3 * 2x + 6 = 24
Your last correct step was: 6x + 6 = 24

Please choose the operation you are attempting:
1: Add Two Terms
2: Subtract Two Terms
3: Multiply Two Terms
4: Distribute Over Parenthesis
5: Move A Term To The Other Side Of The Equation
6: Divide Through By A Term
7: Remove Parenthesis

Q: Give Up On This Problem

Enter Your Choice:
```

Figure 2.6: A Repeated Mistake

The Tutor then attempts to step the student through the operation. For example, if the student chooses to add two terms, the Tutor prompts the student for each term, then prompts the student for the result. The Tutor then checks to see if this step is possible for the given equation by referring to a list of possible mistakes programmed into the Tutor.

[See Table 1] The student is informed if he has made any of these mistakes and is asked to try again.

Table 1: Possible Mistakes

Step	Possible Mistakes
adding two terms	• term(s) is (are) not in equation • terms are not in position to be added • terms are different types (e.g. 2+3x)
subtracting two terms	• term(s) is (are) not in equation • terms are not in position to be subtracted • terms are different types (e.g. 2-3x)
multiplying two terms	• term(s) is (are) not in equation • terms are not in position to be multiplied • multiplication of terms requires exponents (e.g. 2x*3x)
distributing over parenthesis	• term(s) is (are) not in equation • neither term is a parenthetical expression • student attempts to distribute the multiplication of terms in one step • terms are not in position to be distributed • distribution of terms requires exponents [e.g. 2x(3x+1)]
moving a term to the other side of the equation	• term to move is not in equation • term is not in position from which is can be moved to the other side of the equation • attempting to move the opposite of the term
dividing equation through by term	• equation is not evenly divisible by the term
removing parenthesis	• term is not a parenthetical expression • parenthetical expression is not in equation

If the student has not made one of the mistakes in Table 1, the Tutor attempts to diagnose the error by comparing the information it has about the step that the student is trying

to take against the list of common errors in the error file. [See Section 2.2.2] For every match that is found, the corresponding hint is given to the student. If there are no matches, nothing is printed out. [See Figure 2.7] The student is then asked try to enter the result of the operation again.

```
Here are possible errors:

You are only dividing some of 6x + 6 = 24 by 3.
When dividing through, each_term{6x, 6, 24} should be
divided by 3.

2x + 6 = 8: is not right, would you like to try again? (Y or N)
```

Figure 2.7: Diagnosis of the Error

Once the student has entered the correct result, he is asked to enter the equation that results after this operation has been applied. This entire process then repeats as the student continues to step through solving the problem.

2.2 Teacher Subsystem

The teacher also plays an important role in the effectiveness of the Tutor. He is responsible for creating challenging problems files, editing error files, and utilizing feedback from log files. The role of the teacher is summarized in Figure 2.8.

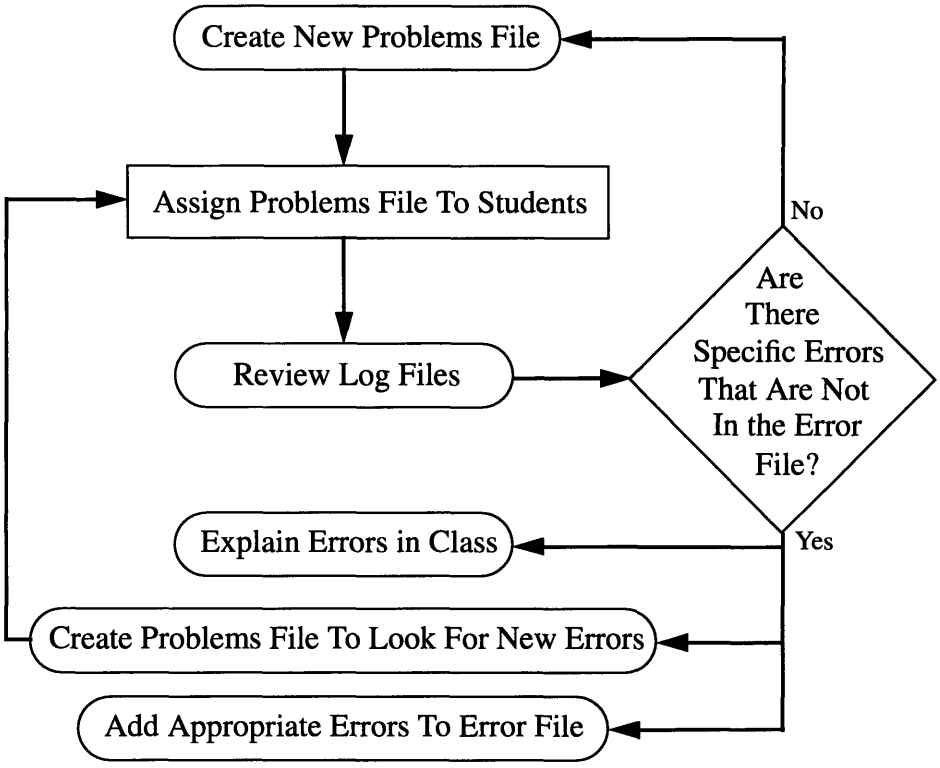


Figure 2.8: Teacher Flow Chart

2.2.1 Problems Files

Problems files contain the examples that form an assignment. [See Figure 2.9] They can be composed using a text editor. Each line of a problem file is a comment, an example or a blank line. The Tutor reads in problems files one at a time. If the line is blank, it is ignored. If the line begins with the character '$', it is considered a comment and is printed to the screen. Otherwise the line is considered to be an equation.

The Tutor does not accept all equations. Equations must be first-order single-variable equations that have integer solutions. Equations that have infinite solutions or no solution are not accepted. If the Tutor tries to load an equation that does not meet these requirements, an error message is output to the screen and the log file; the Tutor then reads in the

next line. The variable of an equation can be any character 'a' through 'z', upper or lower case. Equations can also contain integers and the characters '=', '+', '-', '*', '(' and ')'. Multiplicative expressions and parenthetical expressions may be embedded within themselves and each other.

```
$This is an example of a problems file
$Lines that begin with '$' are comments
$The following blank line will be ignored

$The following problem will not be loaded
$because it has no solution
4x+2=4x+1

$The following problem will not be loaded
$because it has infinite solutions
4x=4x

$The following problem will not be loaded
$because it has a non-itegral solution
3x=7

$The following are legal equations
x=1+1+1+2+2+2
3x+5=5x+9
3(4x+1)-4*1=(5+6)x
2x+2(2x+2(2x+2)-2x)=2(2x+2(2x+2)-2x)+4
```

Figure 2.9: Sample Problems File

2.2.2 Error Files

The most difficult task for a teacher is learning to use the input language for errors. Using this language, teachers can program the Tutor to look for mistakes that their students might make. Teachers can also include corresponding hints to be given to the student when mistakes are made.An error file contains a list of errors. Each error is made up of

three section: the assignment, the error description and the error message. Each section ends with ';'.

2.2.2.1 Assignment

The assignment is the part of the error that attempts to assign the operands of the student's single step operation to the variables in the error description. It is formatted in the following way:

<operation>(<operand1>, <operand2>) => <result>

Each of the possible operations, "add", "sub", "mult", "dist", "move" and "div", corresponds to one of the possible steps a student may take. [See Figure 2.5] If the operation of a particular step matches the student's step, the Tutor will then attempt to assign the student's values to the operands and the result.

The operands and the result are variables called term descriptions. Each term description has two parts, a variable and a type. The variable can be any lower case letter. The type can be '#', 'x', 'X', '()', '*', '?' or 'EQ'. [See Table 2] To successfully complete the assignment, the Tutor attempts to map the student's operands and result to the assignment's operands and result. [See Figure 2.10] For the steps "add" and "mult", the Tutor will attempt to make the assignment of the operands in either order. If the Tutor can successfully assign the student's operation, operands and result to the assignment's variables, a table is created to hold these values. The Tutor then evaluates the error description using this table. If the student's values cannot successfully be assigned to the variables, the Tutor attempts to make an assignment with the next error in the error file.

Table 2: Term Descriptions

type	can represent	examples
#	constants	1, 0, -10
x	variable expressions	x, 3x, -13x
X	variable expressions	x, 3x, -13x
()	parenthetical expressions	(2x+1), (3*(3+4x)), (7)
*	multiplicative expressions	4*3, 2(8x+2), (1+2)(3+1)x
?	constants, variable expressions, parenthetical expressions, multiplicative expressions	5, 4x, (3x+2), 3(5x-4*8), 2((3x-8)6)
EQ	equations	3x+5=6x-7, 2(4x-1)=14

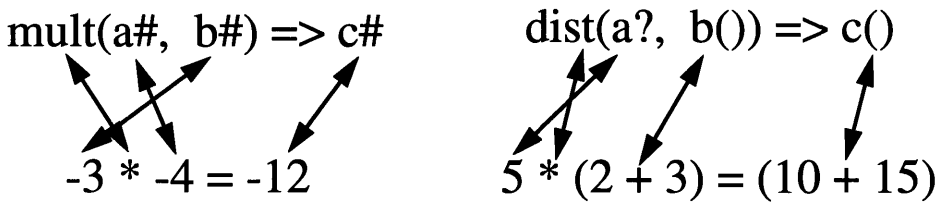
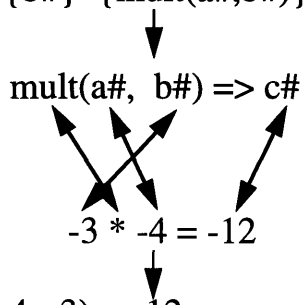


Figure 2.10: Sample Assignments

2.2.2.2 Error Description

Each error description defines the conditions which determine if the student's input is an error. First, assigned variables from the assignment are used as variables for error descriptions. [See Figure 2.11]

```
mult(a#, b#)=>c#
;&(=(c#, negate(mult(a#, b#))),
    &(
       <(a#, 0),
       < (b#, 0)))
;Remember, a negative times a negative is a positive
{a#}*{b#}={mult(a#,b#)};
```




```
mult(-4, -3)=>-12
;&(=(-12, negate(mult(-4, -3))),
    &(
       <(-4, 0),
       < (-3, 0)))
;Remember, a negative times a negative is a positive;
{a#}*{b#}={mult(a#,b#)};
```

Figure 2.11: Error Description Assignment

Operations

Operations can be either binary operations, unary operations, or term descriptions. Binary operations take two operands and return a single result. Unary operations take one operand and also return a single result. [See Table 3] The operands for both types of operations can be either term descriptions from the assignment portion of the error, or results of other operations. [See Figure 2.12]

Binary operations have the form: <operation>(<operand1>, <operand2>)

Unary operations have the form: <operation>(<operand>)

Results of binary and unary operations may represent either a single value or a list of values. List of values can be either and-lists or or-lists. During evaluation, every value of an and-list and only one of the values of an or-list must meet the condition for it to be true.

Table 3: Operations

Operation	Type	Operator	Description
add	binary	add, +	adds two terms, appends term to polynomial, adds term or polynomial to both sides of equation, concatenates polynomials
subtract	binary	sub, -	subtracts second term from first, appends negation of term to polynomial, appends negation of term or polynomial to both sides of equation, concatenates one polynomial with negation of another
multiply	binary	mult, *	multiplies two terms, multiplies polynomial or equation by term or polynomial,
divide	binary	div, /	divides first term by the second, divides a polynomial by a term, divides and equation by a term
mod	binary	mod, //	returns the integer remainder of dividing the first term by the second
and	binary	and, &	returns an and-list of the two operands
or	binary	or, |	returns an or-list of the two operands
remove_term	binary	remove_term	removes a term from a polynomial or an equation
append_term	binary	append_term	appends a term to a polynomial or an equation
min	binary	min	returns the minimum of two terms
max	binary	max	returns the maximum of two terms
make_equation	binary	make_eq	makes two polynomials into an equation

Table 3: Operations

Operation	Type	Operator	Description
every_term	unary	any_term, a_term, some_term, E	returns an and-list of the embedded terms of a term, a polynomial or an equation
any_term	unary	every_term, all_terms, each_term, A	returns an or-list of the embedded terms of a term, a polynomial or an equation
negate	unary	negate, neg, -	negates a term, polynomial or an equation
absolute_value	unary	absolute_value, abs, ||	returns the absolute value of a term
l_poly	unary	l_poly	returns the left polynomial of an equation
r_poly	unary	r_poly	returns the right polynomial of an equation
get_coef	unary	get_coef	returns the coefficient of a variable expression

mult(b?,every_term(a()))

mult(add(negate(a?), b?), a?)

Figure 2.12: Sample Operations

When evaluating operations, the Tutor uses the error description obtained with the values from the assignment table assigned to the variables. It then replaces each operation with the result of applying its operator to its operands. If one of the operands of an operation is itself an operation, the Tutor evaluates the inner operation and uses its result as an operand for the original operation.[See Figure 2.13]

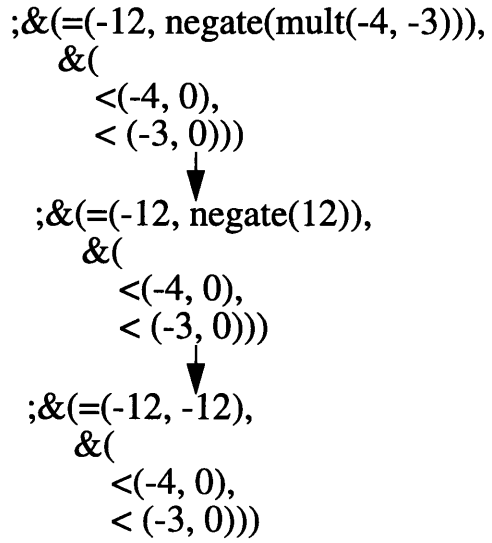


Figure 2.13: Evaluating Operations

Conditions

Conditions are used to compare the results of applying the operations to the variables from the assignment. [See Table 4] Each condition takes two operands, both of which are operations.

Conditions have the form: <condition>(<operand1>, <operand2>)

Table 4: Condition Operators

Condition Operator	Description
=	equal
~=	not equal
<	less than
>	greater than
<=	less than or equal
>=	greater than or equal
~<	not less than
~>	not greater than
~<=	not less than or equal
~>=	not greater than or equal

Conditions can then be combined in pairs using the logic operators "&" (and) and "|" (or). Each logic operation takes two operands, each of which is either a condition or another logic operation.

Condition logic has the form: <logic>(<operand1>, <operand2>)

The Tutor uses the values obtained by evaluating the operations to determine if the conditions have been met in the conditions and condition logic of the error description. [See Figure 2.14]

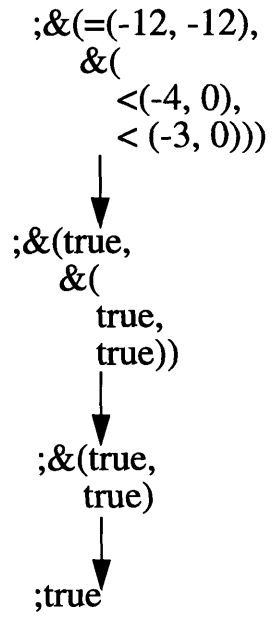


Figure 2.14: Evaluating Conditions and Condition Logic

2.2.2.3 Error Message

An error message is output to the screen if the assignment is possible and the error description evaluates to true. Operations can be embedded in error messages by placing them in braces. The Tutor then outputs the result of evaluating the operation in place of the string between the braces. [See Figure 2.15] If the operation cannot be evaluated, the entire error message is printed out as a string of characters.

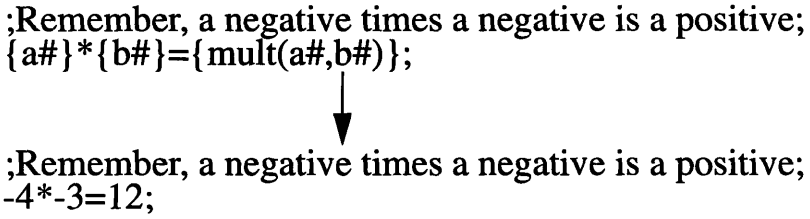


Figure 2.15: Embedded Operations in Error Messages

2.2.3 Log Files

During each session, the Tutor keeps a log of everything that has occurred. This includes all comments, examples, input, error messages, and hints. It is organized in such a manner that it is easier to review than most students' scratch paper. [See Figure 2.16] Students who complete a problems file can turn in their log file after they are finished. The teacher can then review the file to determine if the student has done the problems correctly and also determine where the student is making mistakes. The teacher now has access to the steps that each student has taken to solve each problem. A teacher who carefully reviews his student log files may discover unexpected common mistakes. He can use this knowledge to add appropriate errors to his error files. He can then create new problems files that will test his students in situations where they might make this error. He may also wish to discuss the mistakes that he has noticed with his class.

NEXT EQUATION: 4x = 8
GUESS: x=4
MISTAKE
GUESS: Y
GUESS: x=2
Student Solved the Problem Correctly

NEXT EQUATION: 3 * (4 + 9) = 3x + 6
GUESS: 3(-5)=3x+6
MISTAKE
GUESS: 3(-5)=3x+6
REPEATED MISTAKE
CHOICE: ADD: 4, 9
RESULT: -5
POSSIBLE ERRORS:
You are subtracting instead of adding.
4 + 9 = 13
Try again: 4 + 9 = 13
CORRECT
GUESS: 3 * (13) = 3x + 6
GUESS: 39=3x+6
GUESS: 45=3x
MISTAKE
GUESS: 45=3x
REPEATED MISTAKE
CHOICE: MOVE: 6
RESULT: 39 + 6 = 3x
POSSIBLE ERRORS:
When you move a term to the other side of the equation, you must change its sign. 6 -> -6 when you move it to the other side.
GUESS: 33 = 3x
GUESS: 1221=111x
GUESS: x=11
Student Solved the Problem Correctly

Figure 2.16: Excerpt from Log File

2.3 Equation Equivalence

The most important factor in allowing step-by-step problem solving is creating a flexible equation equivalence function. Since the Tutor must accept any legal simplification of the equation and its solution as input, it was necessary to develop a technique for checking acceptability.

When the Tutor reads in an equation, whether it be from the problems file or the student's input, it generates a subset of possible equivalent states. This subset includes the original equation, all of the possible next steps that can be generated using the single step operations in Figure 2.5 (except dividing through by a term), and at least one path to the answer. [See Figure 2.17] Student's equations are initially checked against this set of equations. If the student's input is the answer, he has completed this example. If the student's input is another equation in this set, the student is on the right track.

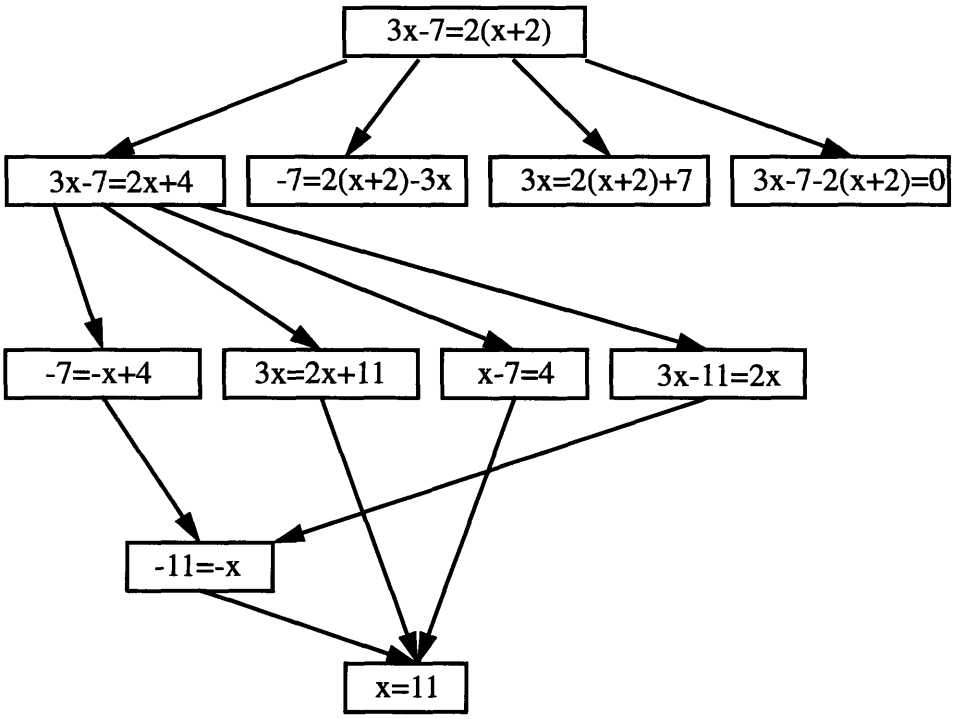


Figure 2.17: Tutor-Generated Subset of Possible States

If the student's input is not in this set, it is still possible that it is a correct step toward the solution. The student's input is evaluated and a set of equations, such as the one created for the original equation, is generated. If the solution to the student's equation is equal to the solution for the original equation, the Tutor accepts this input as a legal step. In this way, the Tutor checks for equivalence rather than equality.

Chapter 3

Results

3.1 Testing

Shaina Garland, an eighth-grade student from Brookline, Massachusetts with approximately one-and-one-half years of algebra experience, completed a set of problems. [See Appendix Figure A.1] The rules file that she used contained twenty-one possible errors. [See Appendix Figure A.2] Her entire session was logged and later examined. [See Appendix Figure A.3] Although it was initially uncomfortable, Shaina was able to solve problems in a step-by-step manner without scratch paper. Her chief complaint was that due to the text-based interface, it was sometimes difficult to locate her last correct step on the screen. Unfortunately for the experiment, Shaina was a very good problem solver and therefore did not make any unintentional mistakes. The only mistake she did make, was made on purpose in order to see how the Tutor would react. This mistake was identified and the Tutor suggested a hint. Although this test of the Tutor did not prove that it was effective in helping to diagnose common mistakes, it did show that it is possible for some students to solve equations with the Tutor without scratch paper.

3.2 Incompleteness

Because of the nature of the problem that the Tutor was designed to solve, many trade-offs were made in its design. These trade-offs resulted in the incompleteness of certain subsystems of the program.

3.2.1 Mathematical Scope

The Tutor was limited to first-order, single-variable algebraic equations with integral solutions. The original design for the Tutor permitted the use of fractions, but this idea was soon dismissed because of time constraints on the implementation. On a more general level, the Tutor does help to show the benefits of the paradigm of giving hints based in the diagnosis of common errors in mathematics. By adding to the existing Tutor, more algebraic rules could be supplied to improve the project's mathematical scope.

3.2.2 Planning Errors

One drawback of the Tutor is that it assumes that students know the correct steps to take when solving a problem. In order for the student to receive a diagnosis of his mistake, he must be able to determine which step to take next in solving the equation. For a student just beginning to learn algebra, this might be very difficult, or perhaps impossible. The Tutor therefore essentially only solves "half" the problem. It is up to the student to determine what steps to take, and the Tutor to determine if he has taken them correctly. A possible solution to this problem is to provide the Tutor with a mechanism that gives the student a hint as to what to do next. [See Section 4.2.3]

3.2.3 Language Limitations

Another area in which the Tutor is incomplete is its error language. [See Section 2.2.2] Because of the diverse nature of algebraic misconceptions, a language that describes all of them would be very large. The Tutor's language was designed to handle many common mistakes, but there are many it does not handle, i.e., the example in section 1.1.1.2:

- Solve: 3x=36
- Student guesses: x=6

could not be described using the Tutor's error language. The main problem results in the fact that the student would most likely not be able to choose the operation he is attempting from the list supplied by the Tutor. This is related to the problem in section 3.2.2 whereby the student does not know the correct step to take next.

Even if the Tutor is able to determine the operation and operands of the student's next step, some errors still cannot be described with the error language. One example of such a bug is described in Brown and Burton's "Diagnostic Models for Procedural Bugs in Basic Mathematical Skills". [1] The error that they describe is one in which the student forgets to reset the "carry register" to zero so that the amount carried is accumulated across the columns. For example, if the problem is 328+917, the student would answer 1345. "...he proceeds as follows: 8+7=15 so he writes 5 and carries 1; 2+1=3 plus the 1 carry is 4; lastly, 3+9=12 but that 1 carry from the first column is still there-it has not reset-so adding it to this column gives 13." Note that this type of error could occur easily if the student is presented with the problem x=328+917. The Tutor is unable to diagnose such a mistake. It would merely notify the student that he has made a mistake and offer no suggestions as to how to fix it.

3.3 Uncertainty

Along with the incompleteness of certain aspects of the Tutor, there is also the issue of uncertainty. When the Tutor thinks it has identified the student's error, it may or may not be correct. It is possible that even though the student's mistake fits the description of a par-

ticular example, the students has actually made an entirely different mistake. For example, the error-language equivalent of the example in section 1.1.1.1:

- Simplify: A(B*C)
- Student knows: A(B+C)=AB+AC
- Student guesses: A(B*C)=AB*AC

might look something like:

dist(a?, b())=>c()
;=(c(), mult(mult(a?, mult(a?, b()))))
;You have assumed that when you multiply a term by a parenthetical expression containing a multiplicative expression that each embedded term of the multiplicative expression is multiplied by the term.;

Note that the result of this operation is still a parenthetical expression because the Tutor considers removing parenthesis an operation in itself. Also note that there is no way to tell if the parenthetical expression contains a multiplicative expression and therefore the error message should be generalized. The resulting message will therefore not be as effective and may even be confusing for the student. It is also possible that the student had actually made an entirely different mistake. For example, if the student is multiplying -1(-2), and he arrives at the answer -2, it probably is not the case that he is multiplying (-1*-1*-2). It is more likely that he has forgotten that a negative times a negative is a positive. If this mistake was also described in the error language, both error messages would be given to the student; but if it was not, only the incorrect message would be given. Error messages should therefore be worded as if they are possible mistakes and not certain mistakes.

Chapter 4

Conclusion

4.1 Summary

The development of the Tutor has helped show that it is possible to create an intelligent algebraic tutor that exploits students' common misconceptions. The Tutor provided an environment in which students are encouraged to solve first-order, single-variable equations at their own pace without scratch paper. It was designed to recognize when a student inputs the correct answer and when the student inputs a step that is on the right track to the answer. By allowing students to work through problems in a step-by-step manner, the Tutor is able to quickly identify in which step the student is making his mistake.

The Tutor can be programmed to diagnose specific misconceptions. It does this by allowing teachers to input common errors in an error language. This language attempts to match students' problems taking single steps in solving a problem to a common misconception. If a match is made, the Tutor then outputs an intelligent hint or suggestion, also constructed by the teacher. Teachers are also able to monitor the work of their students by examining log files that record students' entire sessions with the Tutor.

4.2 Further Work

Before the Tutor can become an effective teaching tool for schools, it needs further development. Although it provides a solid foundation for researching the creation of tutors that exploit

student misconceptions, it is not a finished product. Some suggestions for improvement are listed in this section.

4.2.1 Windows-Based Systems

The interface of the Tutor is entirely text-based. Although this is adequate for research purposes, a windows-based system would be more effective for students. In testing the software, it was noticed that individuals unfamiliar with the program may find it difficult to locate the information that they need. This is due to the fact that in text-based systems, information is constantly scrolling up the screen and out of view. In a windows-based system, these problems can be avoided by providing specified windows to display useful information. For example, there could be a computation history window which displays all of the student's past steps and a pop-up error window that notifies the student that he has made a mistake. Also, a calculator could be provided for the student to compute difficult arithmetic. These enhancements would make the Tutor more appealing and practical.

A windows-based system would also prove to be helpful by allowing students to highlight terms with the mouse. With the existing Tutor, operands for an operation, must entered with the keyboard. This could present ambiguity. For example, if the student is working on the equation x=2+2*3+3, and wants to add the terms 2 and 3, the Tutor would assume that the he is referring to the 3 that is not being multiplied by 2. However, it is possible that the student is actually attempting to add the other 3. The Tutor would not detect this mistake. In a windows-based system, the student could highlight the 3 that he was attempting to add, enabling the Tutor to recognize the error.

4.2.2 Equation Check

Another feature that could be added is an equation check for teachers. Since the Tutor only accepts first-order, single-variable equations with a single integral solution, it is possible that a teacher could enter an unacceptable equation in the problems file. To avoid such an error, teachers currently must load the entire problems file in the pretense that they are a student trying to complete the assignment. An equation checker could be implemented that would allow teachers to quickly determine whether an equation is acceptable.

4.2.3 Hints

The Tutor successfully diagnoses errors that students make when performing step-by-step operations, however it assumes that the student has the ability to determine which steps to take in solving the problems. If the student is not familiar with the legal algebraic operations that can be made on equations, he will not learn significantly from the Tutor. This implies that before it can be a stand-alone teaching tool, the Tutor needs a system to help students choose possible next steps.

One possible way to implement such a system would be to provide the tutor with a hint mechanism that, on request, could suggest possible next steps for the student. Much of the information needed to incorporate such a system depends on information that already exists in the program's data structures. This is due to the fact that whenever the program computes a possible next step, it also specifies which operation was used to compute it. The student would then use this hint mechanism to determine which step to take next. Once the student can take a legal step, the Tutor can then diagnose errors made in executing it.

4.2.4 Generalization

It is important to note that the underlying premises that the Tutor is based upon are not exclusive to algebraic equations. The paradigm of diagnosing common misconceptions in step-by-step problem solving can also be applied to many other mathematical and scientific topics. A similar tutor could be designed to help students solve physics problems or learn to integrate. This conceptual generalization could prove to be very effective in computer tutoring.

The drawback of such a generalization is that the diagnosis of errors in a particular field of study depends on specific rules inherent to that field. The Tutor was designed to handle first-order, single-variable algebraic equations. To implement a different subject would require much change in the underlying structure of the entire program. The general paradigm would need to be reimplemented for each new subject.

4.3 Conclusion

The Tutor provides the groundwork for creating educational software that exploits common student misconceptions to give specific hints. The Tutor provides an environment in which students are able to solve first-order, single-variable algebraic equations in many different ways. By allowing students to solve problems in their own way, the Tutor encourages them to explore different techniques and to find one that is comfortable.

The Tutor also provides an error language which can describe many common student misconceptions. This language allows the Tutor to recognize many student errors when they occur and

respond to them with an appropriate hint. This paradigm could be generalized to other fields of study.

The Tutor also provides a technique for involving the teacher in the educational software process. Teachers are able to carefully monitor their students' work by analyzing the Tutor's log files. Also, for a teacher to enter common errors with the error language, he must attempt to think like his students. He must try to imagine what mistakes his students will be likely to make. This can help to improve his ability to understand why a student might makes a mistake; which, in turn, could help improve his teaching skills.

References

[1] Brown, John Seely and Burton, Richard R., "Diagnostic Models for Procedural Bugs in Basic Mathematical Skills", Cognitive Science, 1978, 2, 155-192.

[2] Matz, M., "Towards a Process Model for High School Algebra Errors" in Sleeman, D. and Brown, J. S., Intelligent Tutoring Systems, Academic Press, Inc., London, England, 1982.

[3] McArthur, David and Stasz, Cathleen, An Intelligent Tutor for Basic Algebra, The RAND Corporation, Santa Monica, CA, 1990.

[4] Miller, Samuel K., Selecting & Implementing Educational Software, Allyn And Bacon, Inc., Boston, MA, 1987.

[5] Peterson, Dale, Intelligent Schoolhouse, Reston Publishing Company, Reston, VA, 1984.

[6] Pride, Mary, Pride's Guide to Educational Software, Crossway Books, Wheaton, IL, 1993.

[7] Walker, Decker F. and Hess, Robert D., Instructional Software, Wadsworth Publishing Company, Belmont, CA, 1984.

[8] Watson, Deryn, Development of CAL: Computers in the Curriculum, Harper & Row Publishers, London, England, 1987.

[9] Wenger Etienne, Artificial Intelligence and Tutoring Systems, Morgan Kaufmann Publishers Inc., Los Altos, CA, 1987.

Appendix A

Test Data

```
$
$Shaina,
$
$Please complete the following problems. I would prefer if you did not use scratch paper.If
$you have any questions, ask me.
$
$-Brian
$
$Problem 1
x=-3*-2
$Problem 2
x=1+2+3+4+5+11
$Problem 3
3x+6=7x-10+0
$Problem 4
2(x+4x)-25=15
$Problem 5
3(x+2x+3x+4x+5x+30)=45
$Problem 6
3(4x*2)+6=2x(3+5)-18
$Problem 7
7x(3+1)-10(2x+3)=-3(-2x+6)
$Problem 8
x=1+3*8+3-2*12
$Problem 9
4*3x+4*16=4*6x+4*1
$Problem 10
24*2+30=16x*3-18
$The following problems might be a little tricky.
$Just take it step by step and you should be OK
$
$Problem 11
2(2(2(1-2)-2)-2)=4x+8
$Problem 12
2x+2(2x+2(2x+2)-2x)=2(2x+2(2x+2)-2x)+4
$
```

```
$Thank you very much for your help.
$
```

Figure A.1: Problems File

```
add(a?, 0) => b?
;~=(b?, a?)
;Whenever you add 0 to a term, the answer is always the term.
{a?} + 0 = {a?};

add(a?, b?) => c?
;=(c?, sub(a?, b?))
;You are subtracting instead of adding.
{a?} + {b?} = {add(a?, b?)};

add(a?, b?) => c?
;=(c?, mult(a?, b?))
;You are multiplying instead of adding.
{a?} + {b?} = {add(a?, b?)};

add(a?, b?) => c?
;=(c?, div(a?, b?))
;You are dividing {a?} by {b?} instead of adding;

add(aX, b#) => c?
;=(c?, add(b#, get_coef(aX)))
;You can't add var_expressions and constants;

sub(a?, 0) => b?
;~=(b?, a?)
;Zero subtracted from any term is always the term.;

sub(0, a?) => b?
;~=(b?, negate(a?))
;Any term subtracted from 0 is always the negative of that term.;

sub(a?, b?) => c?
;=(c?, add(a?, b?))
;You are adding instead of subtracting.
{a?} - {b?} = {sub(a?, b?)};
```

```
mult(aX, bX) => c?
;=(aX, aX)
;Unfortunately, this program is not designed to handle exponents.;

mult(a#, b#) => c#
;&(=(c#, negate(mult(a#, b#))),
 &(
   <(a#, 0),
   <(b#, 0)))
;Your answer of {c#} is the negative of the answer.
Remember, a negative times a negative is a positive.;

mult(a#, b#) => c#
;&(=(c#, negate(mult(a#, b#))),
  &(
   <(a#, 0),
   >(b#, 0)))
;Your answer if {c#} is the opposite of the actual answer.
Remember, a negative times a positive is a negative.;

mult(a#, b#) => c#
;&(=(c#, negate(mult(a#, b#))),
  &(
   >(a#, 0),
   >(b#, 0)))
;Your answer if {c#} is the opposite of the actual answer.
Remember, a positive times a positive is a positive.;

mult(a?, 0) => b?
;~=(b?, 0)
;Zero times any term is always zero.;

mult(a?, b()) => c()
;&(=(every_term(c()),
   or(any_term(b()), mult(a?, any_term(b())))),
  =(any_term(b()), any_term(c())))
;When multiplying any term by a parenthetical expression, each term in parenthesis must
be multiplied by that term.
{every_term(b())} is multiplied by {a?}.
The answer should be {mult(a?, b())}.;

mult(a?, 1) => b?
;~=(b?, a?)
;Whenever you multiply a term by 1, the answer is always the term.
```

```
{a?} * 1 = {a?};

mult(a?, 0) => b?
;~=(b?, 0)
;0 times {a?}, like any term, is always 0.;

mult(a?, b?) => c?
;=(c?, add(a?, b?))
;You are adding instead of multiplying.
It should be {a?} * {b?} = {mult(a?, b?)}.;

dist(a?, b()) => c()
;&(=(every_term(c()),
    or(any_term(b()), mult(a?, any_term(b())))),
  =(any_term(b()), any_term(c())))
;When multiplying any term by a parenthetical expression, each term in parenthesis must
be multiplied by that term.
{every_term(b())} is multiplied by {a?}.
The answer should be {mult(a?, b())}.;

move(aEQ, b?) => cEQ
;|(=(cEQ,
    make_eq(remove_term(l_poly(aEQ), b?),
          append_term(r_poly(aEQ), b?))),
  =(cEQ,
    make_eq(remove_term(r_poly(aEQ), b?),
          append_term(l_poly(aEQ), b?))))
;When you move a term to the other side of the equation, you must
change its sign. {b?} -> {negate(b?)} when you move it to the other side.;

div(aEQ, b?) => cEQ
;&(=(every_term(cEQ),
    or(any_term(aEQ),
      div(any_term(aEQ), b?))),
  =(any_term(aEQ), any_term(cEQ)))
;You are only dividing some of {aEQ} by {b?}.
When dividing through, {every_term(aEQ)} should be
divided by {b?}.;
```

```
div(aEQ, b?) => cEQ
;=(b?, 0)
;You cannot divide by 0;
```

Figure A.2: Error File

```
Shaina Garland

Enter the rules file: ../final/master_rules
Enter the problems file: ./shaina.probs

$
$Shaina,
$
$Please complete the following problems. I would prefer if you did not use scratch paper.
If
$you have any questions, ask me.
$
$-Brian
$
$Problem 1

NEXT EQUATION: 1x = -3 *- 2
GUESS: x=6
Student Solved the Problem Correctly
$Problem 2

NEXT EQUATION: 1x = 1 + 2 + 3 + 4 + 5 + 11
GUESS:

QUIT PROBLEM
$Problem 3

NEXT EQUATION: 3x + 6 = 7x - 10 + 0
GUESS: 3x + 6 = 7x -10 +0
On the right track...
The equation is: 3x + 6 = 7x - 10 + 0
Your last step was: 3x + 6 = 7x - 10 + 0
GUESS: 6 = 4x -10
On the right track...
The equation is: 3x + 6 = 7x - 10 + 0
Your last step was: 6 = 4x - 10
```

GUESS: 16 = 4x
On the right track...
The equation is: 3x + 6 = 7x - 10 + 0
Your last step was: 16 = 4x
GUESS: x =4
Student Solved the Problem Correctly
$Problem 4

NEXT EQUATION: 2 * (1x + 4x) - 25 = 15
GUESS: 2x + 8x -25 = 15
On the right track...
The equation is: 2 * (1x + 4x) - 25 = 15
Your last step was: 2x + 8x - 25 = 15
GUESS: 10x -25 = 15
On the right track...
The equation is: 2 * (1x + 4x) - 25 = 15
Your last step was: 10x - 25 = 15
GUESS: 10x =40
On the right track...
The equation is: 2 * (1x + 4x) - 25 = 15
Your last step was: 10x = 40
GUESS: x=4
Student Solved the Problem Correctly
$Problem 5

NEXT EQUATION: 3 * (1x + 2x + 3x + 4x + 5x + 30) = 45
GUESS: 45x +90 = 45
On the right track...
The equation is: 3 * (1x + 2x + 3x + 4x + 5x + 30) = 45
Your last step was: 45x + 90 = 45
GUESS: 45x = -45
On the right track...
The equation is: 3 * (1x + 2x + 3x + 4x + 5x + 30) = 45
Your last step was: 45x = -45
GUESS: x = -1
Student Solved the Problem Correctly
$Problem 6

NEXT EQUATION: 3 * (4x * 2) + 6 = 2x * (3 + 5) - 18
GUESS: 12x * 6 +6 = 6x + 10x -18
MISTAKE
GUESS: y
Student understands error.
The equation is: 3 * (4x * 2) + 6 = 2x * (3 + 5) - 18

Your last step was: 3 * (4x * 2) + 6 = 2x * (3 + 5) - 18
GUESS: 24x +6 = 6x+10x -18
On the right track...
The equation is: 3 * (4x * 2) + 6 = 2x * (3 + 5) - 18
Your last step was: 24x + 6 = 6x + 10x - 18
GUESS: 24x +6 = 16x =10
Bad Format: Too many =’s in equation
Try again? y
The equation is: 3 * (4x * 2) + 6 = 2x * (3 + 5) - 18
Your last step was: 24x + 6 = 6x + 10x - 18
GUESS: 24x +6 = 16x-10
MISTAKE
GUESS: y
Student understands error.
The equation is: 3 * (4x * 2) + 6 = 2x * (3 + 5) - 18
Your last step was: 24x + 6 = 6x + 10x - 18
GUESS: 24x +6 = 10x -18
MISTAKE
GUESS: y
Student understands error.
The equation is: 3 * (4x * 2) + 6 = 2x * (3 + 5) - 18
Your last step was: 24x + 6 = 6x + 10x - 18
GUESS: 24x +6 = 16x -18
On the right track...
The equation is: 3 * (4x * 2) + 6 = 2x * (3 + 5) - 18
Your last step was: 24x + 6 = 16x - 18
GUESS: 8x +6 = -18
On the right track...
The equation is: 3 * (4x * 2) + 6 = 2x * (3 + 5) - 18
Your last step was: 8x + 6 = -18
GUESS: 8x = -24
On the right track...
The equation is: 3 * (4x * 2) + 6 = 2x * (3 + 5) - 18
Your last step was: 8x = -24
GUESS: x = -3
Student Solved the Problem Correctly
$Problem 7

NEXT EQUATION: 7x * (3 + 1) - 10 * (2x + 3) = -3 * (-2x + 6)
GUESS: 28x -20x -30 = 6x -18
On the right track...
The equation is: 7x * (3 + 1) - 10 * (2x + 3) = -3 * (-2x + 6)
Your last step was: 28x - 20x - 30 = 6x - 18
GUESS: 8x -30 = 6x -18

On the right track...
The equation is: 7x * (3 + 1) - 10 * (2x + 3) = -3 * (-2x + 6)
Your last step was: 8x - 30 = 6x - 18
GUESS: 8x -48 = 6x
MISTAKE
GUESS: 8x -48 = 6x
REPEATED MISTAKE
CHOICE: MOVE: -18
RESULT: 8x - 48 = 6x
POSSIBLE ERRORS:
GUESS: 8x - 30 - 18 = 6x
POSSIBLE ERRORS:
When you move a term to the other side of the equation, you must change its sign. -18 -> 18 when you move it to the other side.
GUESS: 2x - 30 = -18
On the right track...
The equation is: 7x * (3 + 1) - 10 * (2x + 3) = -3 * (-2x + 6)
Your last step was: 2x - 30 = -18
GUESS: 2x = 12
On the right track...
The equation is: 7x * (3 + 1) - 10 * (2x + 3) = -3 * (-2x + 6)
Your last step was: 2x = 12
GUESS: x=6
Student Solved the Problem Correctly
$Problem 8

NEXT EQUATION: 1x = 1 + 3 * 8 + 3 - 2 * 12
GUESS: x = 1 +24 -21
On the right track...
The equation is: 1x = 1 + 3 * 8 + 3 - 2 * 12
Your last step was: 1x = 1 + 24 - 21
GUESS: x=4
Student Solved the Problem Correctly
$Problem 9

NEXT EQUATION: 4 * 3x + 4 * 16 = 4 * 6x + 4 * 1
GUESS: 12x + 64 = 24x =4
Bad Format: Too many ='s in equation
Try again? y
The equation is: 4 * 3x + 4 * 16 = 4 * 6x + 4 * 1
Your last step was: 4 * 3x + 4 * 16 = 4 * 6x + 4 * 1
GUESS: 12x+24=24x +4
MISTAKE
GUESS: y

Student understands error.
The equation is: 4 * 3x + 4 * 16 = 4 * 6x + 4 * 1
Your last step was: 4 * 3x + 4 * 16 = 4 * 6x + 4 * 1
GUESS: 12x +64 =24x +4
On the right track...
The equation is: 4 * 3x + 4 * 16 = 4 * 6x + 4 * 1
Your last step was: 12x + 64 = 24x + 4
GUESS: 64 = 12x +4
On the right track...
The equation is: 4 * 3x + 4 * 16 = 4 * 6x + 4 * 1
Your last step was: 64 = 12x + 4
GUESS: 60 = 12x
On the right track...
The equation is: 4 * 3x + 4 * 16 = 4 * 6x + 4 * 1
Your last step was: 60 = 12x
GUESS: x =5
Student Solved the Problem Correctly
$Problem 10

NEXT EQUATION: 24 * 2 + 30 = 16x * 3 - 18
GUESS: 78=48x-18
On the right track...
The equation is: 24 * 2 + 30 = 16x * 3 - 18
Your last step was: 78 = 48x - 18
GUESS: 96 = 48x
On the right track...
The equation is: 24 * 2 + 30 = 16x * 3 - 18
Your last step was: 96 = 48x
GUESS: x=2
Student Solved the Problem Correctly
$The following problems might be a little tricky.
$Just take it step by step and you should be OK
$
$Problem 11

NEXT EQUATION: 2 * (2 * (2 * (1 - 2) - 2) - 2) = 4x + 8
GUESS: -20 = 4x +8
On the right track...
The equation is: 2 * (2 * (2 * (1 - 2) - 2) - 2) = 4x + 8
Your last step was: -20 = 4x + 8
GUESS: -28 = 4x
On the right track...
The equation is: 2 * (2 * (2 * (1 - 2) - 2) - 2) = 4x + 8
Your last step was: -28 = 4x

GUESS: x = -7
Student Solved the Problem Correctly
$Problem 12

NEXT EQUATION: 2x + 2 * (2x + 2 * (2x + 2) - 2x) = 2 * (2x + 2 * (2x + 2) - 2x) + 4
GUESS: 10x +8 =8x +12
On the right track...
The equation is: 2x + 2 * (2x + 2 * (2x + 2) - 2x) = 2 * (2x + 2 * (2x + 2) - 2x) + 4
Your last step was: 10x + 8 = 8x + 12
GUESS: x=2
Student Solved the Problem Correctly
$
$Thank you very much for your help.
$

QUIT

Figure A.3: Log File

Appendix B

Sample Error Evaluations

The following two sections will depict how the Tutor evaluates errors. The first error determines if the student made a mistake when multiplying a term by zero. The second error ensures that the student distributes a term over every term in parenthesis.

B.1 Multiplying by Zero

In this example, the student is making the mistake 0*5=5. The error checks to see if the student has multiplied a term by zero and got a result other than zero. Since this is true, the error message is printed to the screen.

Error:

```
mult(a?, 0) => b?
;~=(b?, 0)
;0 times {a?}, like any term, is 0.;
```

Student input:

```
Enter your choice: 3 (Multiply Two Terms)
Enter one term: 0
Enter the other term: 5
Enter the result of multiplying the terms: 5
```

Assignment:

```
0 <--> 0
a? <--> 5
b? <--> 5
```

Error Description:

Apply assignment:

~(5, 0)

Evaluate condition:

true

Error Message:

0 times 5, like any term, is 0

B.2 Distributing Over Parenthesis

In this example, the student is making the mistake 2(3+5)=(6+5). He is only distributing the term 2 over one of the two terms of the parenthetical expression. The error checks if every term of the resulting parenthetical expression is equal to either a term of the multiplied parenthetical expression or the distributed term times a term of the multiplied parenthetical expression. It also checks if at least one term of the resulting parenthetical expression is equal to some term of the multiplied parenthetical expression. Although there are exceptions to this error, it will work for most distributions.

Error:

```
dist(a?, b()) => c()
;&(=(every_term(c(), or(any_term(b()), mult(a?, any_term(b()))))),
   =(any_term(b()), any_term(c())))
;When multiplying any term by a parenthetical expression, each term in parenthesis
must be multiplied by that term.
{every_term(b())} is multiplied by {a?}.
The answer should be {mult(a?, b())};
```

Student input:

```
Enter your choice: 4 (Distribute Over Parenthesis)
Enter the parenthetical expression (including parenthesis): (3+5)
Enter the term to distribute: 2
Enter the result (including parenthesis): (6+5)
```

Assignment:

a?<--> 2

b() <--> (3+5)

c() <--> (6+5)

Error Description:

Apply assignment:

&(=(every_term((6+5), or(any_term((3+5)), mult(2, any_term((3+5))))),
=(any_term((3+5)), any_term(c(6+5))))

Apply any_term and every_term:

&(=({6 and 5}, or({3 or 5}, mult(2, {3 or 5}))),
=({3 or 5}, {6 or 5}))

Apply multiplication:

&(=({6 and 5}, or({3 or 5}, {6 or 10})),
=({3 or 5}, {6 or 5}))

Apply or:

&(=({6 and 5}, {3 or 5 or 6 or 10}),
=({3 or 5}, {6 or 5}))

Check conditions:

&(true, true)

Apply &:

true

Error Message:

When multiplying any term by a parenthetical expression, each term in parenthesis must be multiplied by that term.

each_term{3, 5} is multiplied by 2.

The answer should be (6+10)