

**Constructing an Authoring Tool for Intelligent Tutoring Systems with  
Hierarchical Domain Models**

by  
Vilmos Csizmadia

A Thesis  
submitted to the Faculty  
of the  
WORCESTER POLYTECHNIC INSTITUTE  
in partial fulfillment of the requirements for the  
Degree of Master of Science in  
Computer Science

by

---

December, 2003

APPROVED:

---

Dr. Neil T. Heffernan, Thesis Advisor

---

Dr. David C. Brown, Reader

---

Dr. Michael A. Gennert, Department Head

## **Abstract**

Intelligent Tutoring Systems (ITSs), while effective in enhancing students' problem solving skills, are difficult and time-consuming to build. In order to reduce the length and the complexity of ITS construction, authoring tools are used. These tools provide a solid foundation for creating pedagogical exercises for students, and offer graphical user interfaces that eliminate the need for programming expertise. One of the major problems with today's authoring tools is that they are still quite intricate and time-consuming to utilize, even for users who are familiar with them. Their steep learning curves often intimidate users who are only interested in creating simple tutoring systems.

I have designed and implemented an authoring tool, called Mason, which strips away the visual interface design features of today's top ITSs, and focuses on the creation of sophisticated pedagogical exercises using a hierarchical domain model. The exercise creation process includes the definition of numerous components, such as: a problem statement, the desired answer to the exercise, the strategies for tutoring students on the mistakes they make while trying to formulate the correct answer, and diagnostic rules for launching the appropriate strategies for specific student errors.

The ultimate goal of Mason is to be able to significantly reduce the time needed to author text-based ITSs that are able to diagnose student answers and generate pedagogical dialogue accordingly. This goal was verified by using Mason to replicate the architecture of Ms. Lindquist, a sophisticated ITS for algebra that originally took over a year a construct. The replica was finished in less than a week, and was able to emulate Ms. Lindquist's dialogue generation accurately with minor limitations.

# Table of Contents

|   |    |
|---|----|
| Abstract.....   | i  |
| Table of Contents.....  | ii |
| List of Figures.....  | iv |
| Introduction.....   | 1  |
| Chapter 1: Related Work .....   | 4  |
| 1.1 A Brief Introduction to Intelligent Tutoring Systems.....         | 5  |
| 1.2 RIDES.....  | 7  |
| 1.3 Macromedia Authorware .....                                       | 8  |
| 1.4 Eon .....   | 9  |
| 1.5 CMAT.....   | 10 |
| 1.6 Ms. Lindquist, an Intelligent Tutoring System for Algebra.....    | 11 |
| 1.6.1 Dynamic Scaffolding.....  | 12 |
| 1.6.2 Strategy 1: Concrete Articulation .....                         | 14 |
| 1.6.3 Strategy 2: Explain in English First.....                       | 15 |
| 1.6.4 Strategy 3: Convert the Problem into an Example to Explain..... | 17 |
| 1.6.5 Strategy 4: Introduce a New Variable.....                       | 18 |
| 1.7 Chapter Conclusions.....  | 19 |
| Chapter 2: Mason’s Architecture.....                                  | 21 |
| 2.1 The Diagnostic System.....  | 23 |
| 2.2 The Domain Model.....   | 31 |
| 2.2.1 Problems, Blocks, and Answer Trees .....                        | 33 |
| 2.2.2 Questions, Templates, and Answer Rules.....                     | 42 |
| 2.2.3 Strategies and Strategy Sets.....                               | 49 |
| 2.2.4 Diagnostic Rules .....  | 52 |
| 2.3 Chapter Conclusions.....  | 54 |
| Chapter 3: Evaluation .....   | 56 |
| 3.1 The Reconstruction of Ms. Lindquist .....                         | 57 |
| 3.2 Evaluation of Reconstruction Time .....                           | 71 |
| 3.3 Evaluation of Reconstruction Accuracy .....                       | 73 |

|                                       |    |
|---------------------------------------|----|
| 3.4 Chapter Conclusion.....           | 76 |
| Chapter 4: Discussion .....           | 77 |
| 4.1 Limitations .....                 | 77 |
| 4.2 Future Work and Conclusions ..... | 79 |
| References.....                       | 81 |

## List of Figures

|   |    |
|---|----|
| Figure 1 - An example of the RIDES authoring tool in action .....           | 7  |
| Figure 2 - The graphical user interface of the Evaluator tab .....          | 21 |
| Figure 3 - An assembled student answer in Mason .....                       | 22 |
| Figure 4 - An answer tree mapped onto a boolean tree .....                  | 24 |
| Figure 5 - A boolean tree where only the root is a faulty node .....        | 27 |
| Figure 6 - A boolean tree with three faulty nodes .....                     | 27 |
| Figure 7 - A boolean tree indicating that the problem has been solved.....  | 30 |
| Figure 8 - The graphical user interface of the Problems tab.....            | 33 |
| Figure 9 - The graphical user interface of the Answer tab .....             | 34 |
| Figure 10 - The tree representation of an expression.....                   | 35 |
| Figure 11 - The tree structure of the equation with empty nodes .....       | 36 |
| Figure 12 - Binding blocks to the answer tree .....                         | 38 |
| Figure 13 - The complete answer tree.....                                   | 39 |
| Figure 14 - The graphical user interface of the Blocks tab.....             | 41 |
| Figure 15 - The graphical user interface for the Question tab .....         | 42 |
| Figure 16 - The graphical user interface for the Answer Rules tab .....     | 46 |
| Figure 17 - The graphical user interface for the Strategies tab.....        | 51 |
| Figure 18 - The graphical user interface for the Diagnostic Rules tab ..... | 52 |
| Figure 19 - Answer tree for the "Shoe" problem .....                        | 78 |

# Introduction

Well-designed Intelligent Tutoring Systems (ITSs) can be far superior to classroom learning in enhancing students' problem solving skills (Koedinger et. al. 1995). This is mainly due to the fact that both the content and the style of individualized tutoring can be tailored to meet the needs of the situation (Bloom, 1984). An advantageous strategy of ITSs is to observe how individual students solve, or attempt to solve, a set of test exercises, pinpoint problematic areas, and focus on improving the problem solving skills in those areas only. Students learn from their mistakes and build knowledge in an individualized manner (Bruner, 1966; Ginsburg & Opper, 1979).

Although ITSs can be an excellent supplement to class lectures, they are extremely complicated and time-consuming to construct. Without using authoring tools, over 200 hours of ITS development time may be necessary to assemble an hour of instructional material (Woolf & Cunningham, 1987). Most systems need to be constructed from scratch due to their subject specificity (Virvou & Moundridou, 2001). For example, the construction of a system used for training radar technicians in the United States Air Force would differ greatly from the construction of an algebra tutor. Also, the building of a sophisticated ITS requires programming experts and even professional tutors for designing effective pedagogical material.

The objective of this thesis is to create the foundation for a powerful yet user-friendly ITS authoring tool, called Mason. Mason strips away the user interface design features of today's top authoring tools, since the thesis focuses on building a flexible and domain-independent authoring tool for creating pedagogical exercises for students. The

system supports a chat-room-like interface, where only text is exchanged between the student and the tutor.

A standard tutoring system usually has two main components; these are: the **knowledge base** and the **teaching strategies**. The knowledge base contains all the instructional material, while the teaching strategies define the tutoring process (Murray, 1998). These two components can actually be represented as a small network of four ITS elements, which is responsible for all the tutoring procedures. The network consists of a **domain model**, a set of **tutorial strategies**, a **student model**, and a **learning environment**. A domain model contains all the problems, hints, and methods for solving the problems. Tutorial strategies contain the sets of rules the ITS uses to determine what hints or questions should be generated for the student. The student model often keeps track of the student's progress, and pinpoints topics the student finds difficult to understand. Finally, the learning environment provides the student with a user-friendly interface. These four elements are discussed in more detail in the following chapter.

All ITSs use some variation of these four components. For example, Heffernan's Ms. Lindquist uses the student model to diagnose student answers, and, with the help of tutoring strategies, it generates pedagogical dialogue using the tutorial model. Ms. Lindquist's learning environment is also a chat-room-like interface, much like Mason's (Heffernan, 2001). Mason will slightly simplify these four components in order to allow for faster ITS construction. The construction process consists of defining numerous components, such as: problem structures (consisting of problem statements and the desired answers for them), question templates for the strategies that generate pedagogical dialogue for tutoring students, and diagnostic rules for launching the appropriate

strategies for specific student errors. All of these components are organized in a hierarchical fashion.

For the evaluation of this thesis, Mason was used to reproduce the architecture of Ms. Lindquist, an ITS for algebra. Replicating an already existing ITS is a logical way of evaluating an authoring tool, since the differences between the ITS and its replica can be observed by analyzing the dialogue the two systems generate. The construction time of the original ITS and its replica can also be compared in order to conclude whether or not the usage of the authoring tool can shorten the construction process.

The first chapter discusses some of today's most popular authoring tools, their advantages, and their shortcomings. This should provide the reader with a good sense of what solutions others have developed for reducing ITS construction times. Ms. Lindquist is also introduced in detail; the chapter pinpoints the tutoring system's most important features, which need to be emulated by Mason for the evaluation process. The second chapter explains Mason's hierarchical architecture in detail. The goal of this thesis is to create an authoring tool that can significantly reduce the time needed to construct ITSs. Thus, the third and final chapter presents the reader with the evaluation process that illustrates that this goal has been met. It took roughly a year to build Ms. Lindquist from scratch without an authoring tool; using Mason, it took less than a week.



## Chapter 1: Related Work

Many authoring tools for Intelligent Tutoring Systems have been developed in the past that are still in use today; these include: the Learn, Explore, and Practice (LEAP) system (Sparks et. al. 1999), RIDES (Monroe, 1995), and REDEEM (Major et. al. 1977). As we take a look at the newer designs, we can observe that authoring tools are evolving into **domain-independent** systems that place a strong emphasis on graphical user interfaces. A domain-independent authoring tool can construct ITSs for, for example, teaching algebra or training radar technicians. They aren't bound to a specific field of study. The most recent tools allow authors to fully build and customize the student interface; this, of course, can add to the overall ITS development time.

First, this chapter provides the reader with a general introduction to Intelligent Tutoring Systems. It's important to understand the architecture of ITSs before discussing authoring tools for them. Murray (1999) wrote a review that encompassed over twenty authoring tools. The tools, however, are used for research purposes only; there are no commercially available ones today. Thus, the chapter presents a general overview of some of today's top authoring tools. The mentioned authoring tools would not be a logical choice for producing a sophisticated ITS such as Ms. Lindquist due to their specialized system architecture. Being able to closely emulate the algebra tutor's powerful dialogue generation features is one of Mason's primary strengths. The chapter wraps up by discussing Ms. Lindquist and its most important architectural components.

## **1.1 A Brief Introduction to Intelligent Tutoring Systems**

Before we jump into the slowly expanding world of ITS authoring tools, let's take a quick look at tutoring systems. After all, in order to construct ITSs, we need to be familiar with their general architecture. The introduction, in a quick overview, explained that a standard tutoring system usually consists of four main elements: a domain model, a set of tutorial strategies, a student model, and a learning environment. These elements represent a knowledge base, which stores all the instructional material, and teaching strategies that define the tutoring process (Murray, 1998). All ITSs out there follow this structure in some shape or form; although, they might attach different technical terms to the elements mentioned above.

First, there needs to be a model for storing all the pedagogical material of the system. This is the purpose of the domain model; it contains all the problems, hints, templates, and even methods for solving the problems. The second ITS element, tutorial strategies, contains the sets of rules the system uses to determine what hints or questions should be displayed for the student. Some tutoring systems simply merge the tutorial strategies with the domain model; after all, both elements are used for storing system data.

The student model, which is the third element in the list, keeps track of the student progress, and pinpoints the areas that prove to be problematic for the student. For some ITSs, this is a very simple model that only keeps track of which problem the student is currently working on. Currently, Mason doesn't support the production of complex student models that keep track of the student's progress. The last element, the learning environment, provides the student with a user-friendly interface. Modern ITSs, like Macromedia's Authorware, even include sound files, graphics, and graphical user

interface components in their learning environments. Mason provides a basic user interface reminiscent of a chat room, where only text is exchanged between the student and the tutoring system.

Ms. Lindquist, more or less, follows the standard ITS architecture. It expands upon the student model by enhancing it with a set of over seventy production rules. These rules are used by Ms. Lindquist's **model-tracing** system for diagnosing the student answers. The production rules are utilized to generate arithmetically correct solutions to problems, using defined sub-expressions such as variables and numbers. Model-tracing compares these generated solutions to the student's answer, and, using a set of "buggy" production rules, attempts to understand and categorize the student's errors. This is an effective method for not only detecting errors student answers, but also for finding the correct parts. The student model also contains the working memory, which keeps track of the current state of the system.

Finally, the Ms. Lindquist architecture also contains the tutorial model, which generates the dialogue for the student. Dialogue generation is accomplished using four strategies, which are simply an ordered collection of question templates. Since templates and rules for dialogue generation are considered to be system data, it's safe to say that the tutorial model is really just an extension of the domain model. All of these ITS elements, and how Mason can be used to construct them, will be discussed in the following chapters.

As we can see, Ms. Lindquist uses a combination of the four general ITS components. All tutoring systems need to store their pedagogical data and diagnostic

system architecture somehow. The following sections will illustrate what solutions others have developed in order to author these ITS components.

## 1.2 RIDES

RIDES is an authoring tool for composing and delivering graphical simulations and simulation-based training. (Munro et. al. 1977). The research project was funded by the Office of Naval Research and United States Air Force. Using RIDES, the author can take a set of simulations, and turn them into interactive, graphical tutorials. In fact, the reason why RIDES could never be used for reproducing Ms. Lindquist is because of its limitation to produce tutors based on simulations only.

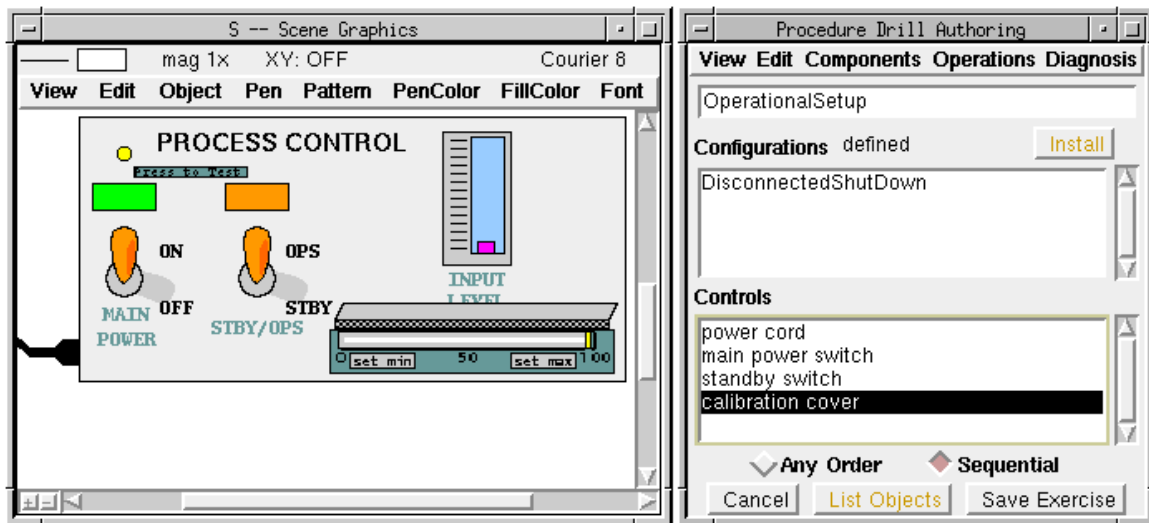


Figure 1 - An example of the RIDES authoring tool in action

The most important feature of RIDES is its human-computer interface (HCI) innovations (see Figure 1). These include: support for constraint and event authoring, which refers to binding special events to the graphical interface, and allowing the author to create procedure tutorials by interactive demonstration. RIDES offer several time-saving features, such as reusable objects and components, which add to the flexibility of

the system (Munro et. al. 1977). Its development model is moderately straight-forward and well organized; this is also advantageous, especially for complex ITSs. The construction process is as follows:

- The author initially specifies the learning objectives, and then determines the required lessons for achieving them.
- The interactive graphical model containing simulations is then created and tested to suit the specifications.
- Lessons are added in the context of the simulations.
- Finally, the lessons are linked to the learning objectives.

Although RIDES is powerful and easy to use, its tutoring process is limited to device component identification, operation, and trouble-shooting (Murray, 1999). The system is mostly used for enhancing procedural skills. Also, the fact that only simulation-based ITSs can be built with RIDES makes the tool an irrational choice for reproducing Ms. Lindquist.

### **1.3 Macromedia Authorware**

Murray (1998) argues that while most common off-the-shelf software (COTS) authoring tools provide unique ITS-construction methods and can be used to construct visually appealing learning environments, they lack sophistication. Authorware, a COTS multimedia authoring tool, exemplifies this issue. Authorware includes a simple demo, which tutors the student on the usage of a camera. The tutoring session is similar to an interactive Microsoft PowerPoint presentation; the tutor asks a set of questions, which the student answers by clicking on the appropriate camera parts. Multiple choice questions are also demonstrated.

The student interface consists of several interactive graphical components. For example, each part of the camera is represented as a separate object. This object-oriented approach provides great flexibility, especially since Authorware supports scripting in multiple languages such as JavaScript (Wilson & Thornton, 2001). One of Authorware's key features is the ability to update the ITS-in-development during its testing phase. While the ITS is running in simulation mode, the user can simply pause the process, make the desired changes to the ITS's structure, and then resume the simulation without having to start all over. Murray points out that although this feature is useful, Authorware lacks modularization and reusability (Murray, 1998).

Scripting support can add great flexibility to an authoring tool, yet the e-learning systems produced are reminiscent of interactive presentations. Authorware lacks the sophistication for reproducing an ITS such as Ms. Lindquist. The authoring tool has a respectable user base that includes American Airlines and Mercedes-Benz, but these corporations utilize Authorware for producing interactive instructional and training material instead of ITSs.

## **1.4 Eon**

The term modularized is what best describes Eon, Murray's own authoring tool (Murray, 1998). Eon allows the user to take the four main ITS components, which are the domain knowledge, the student model, the tutorial strategies, and the student learning environment, and fully customize them (Murray, 1998). The system has an organized methodology for ITS construction. The author starts off by mapping out a topic network, in which the types of links and topic nodes allowed, along with the respective properties, are defined by the topic ontology. Topic types can include facts, concept, principle, etc. A

topic network is constructed using the previously defined links and topics. Relationships are defined among the topics. For complex ITSs, the network editor can get quite elaborate, but Eon's graphical representation makes it readable.

Next, the learning environment is created using Eon's editor, which fully supports graphical user interface design by using widgets. Widgets, which can be pictures, sliders, movies, buttons, text fields, and other objects, are reusable; this is a very useful feature, especially because the widgets are equipped with numerous configurable properties. Next, the student model is defined and teaching strategies are defined. The strategies relate back to the topic network during the tutoring process. In sum, Eon's architecture is a huge network of objects, which adds reusability and great flexibility to the system.

One of Eon's major problems is the fact that it was implemented in SK8, a high-level programming language developed by Apple Computer's Advanced Technology Group (Murray, 1998). The group discontinued the language, leaving it relatively slow and buggy. Eon, however, is still a solid example of modularization. Mason's object-oriented architecture has similar advantages; the customization of objects adds a significant amount of flexibility to the system. Reusability, however, is not supported in the current version of the system, but is only planned for a future version.

## **1.5 CMAT**

Currently in development is the Cognitive Modeling Authoring Tools (CMAT) Suite, which is an advanced production system (Heffernan, 2002). CMAT's unique features include an intelligent GUI builder, which can construct any graphical user interface combinations from standard Java widgets, and a behavior recorder, with which the user can record alternate paths for solving a particular problem. The system also

supports model-tracing. As described previously, model-tracing is a powerful technique in which the system generates all possible correct and incorrect solutions to the current problem. Then, the system checks if the student's answer matches any of the generated solutions. If a match is found, then the student is tutored accordingly.

These solutions can be defined as paths in CMAT. For example, in a tutoring system for adding numbers, the author can specify all the possible student errors as paths. Then, tutoring procedures can be linked to those paths. Mason will have the ability to do partial model-tracing, since it can deduce which portions of the student's answer are correct and which ones are not. This is discussed in the next chapter.

### ***1.6 Ms. Lindquist, an Intelligent Tutoring System for Algebra***

The best method for verifying that Mason, does in fact reduce ITS construction time is by using it to build a copy of an already existing, sophisticated ITS. The goal of this evaluation process is to reproduce Ms. Lindquist's architecture as closely as possible. This requires the dissection of the ITS and the singling out of its most essential features. This chapter describes these features and explains the reasons behind their selection. The current version of Mason is unable to produce an ITS that can completely emulate Ms. Lindquist, since Ms. Lindquist has a general understanding of algebra. This is made possible by a set of over seventy rules, which help the ITS evaluate algebraic expressions and apply mathematical concepts such as the associative and distributive properties. Currently, it's not possible to incorporate such rules into Mason's diagnostic engine.

Yet, even with a lack of understanding of complex algebraic concepts, Mason can create ITSs that can nearly mirror Ms. Lindquist's capabilities. This is due to its dialogue generation system. Ms. Lindquist's strength lies in diagnosing the student's answer and



generating positive, negative, and helpful feedback that will eventually lead the student to the correct answer. In order to achieve this, it uses **dynamic scaffolding** to diagnose the student's answer and find its problematic areas, and a set of four **strategies** to approach those areas from different angles. In order to understand this, it's best to take a look at actual dialogue produced by Ms. Lindquist.

First, this section discusses the architecture of Ms. Lindquist, including some important concepts such as dynamic scaffolding and model-tracing. Then, the four strategies are described in detail; the strategies are the driving force behind Ms. Lindquist's dialog generation. Finally, the strategies' relationship with the rest of the architectural components is explained.

### 1.6.1 Dynamic Scaffolding

When a student inputs an answer, Ms. Lindquist uses a three-step process to generate helpful dialogue based on the student's answer. The system (i) diagnoses the student's current answer and pushes all of its incorrect portions onto a stack, (ii) generates positive feedback for each portion the student got correct, and (iii) focuses the dialog on the incorrect portions that have been pushed onto the stack. This three-step process is called dynamic scaffolding (Heffernan, 2001). In contrast, **static scaffolding** determines the list of questions for the student ahead of time, and does not update the list dynamically. The following sample dialogue shows dynamic scaffolding in action:

**Problem:** Anne is rowing a boat in a lake and is 800 yards from the dock from which she started. She rows back towards the dock at 40 yards per minute for  $m$  minutes and stops to rest. How far is she from the dock now?

**Tutor:** Hello. Please write an expression for the distance Anne has left to row.

**Student:**  $600 - 40 * m$

**Tutor:** No. But,  $40 * m$  is correct for the distance rowed so far. We will come back to the distance Anne has left to row. Let me try to break this down for you. What is her initial distance from the dock?

**Student:** 800

**Tutor:** Correct. Now, let's try this question again: Please write an expression for the distance Anne has left to row.

**Student:**  $800 - 40 * m$

**Tutor:** Correct. You solved the entire problem.

This is the complete dialogue generated for one of Ms. Lindquist's problems. It illustrates several important aspects. First, the user interface is similar to that of a chat program, consisting of nothing more than an input and an output field. The ITS asks a question, and the student inputs the answer. This makes Ms. Lindquist an ideal ITS for Mason to reproduce as there are no complex graphical user interface components to emulate. Second, Ms. Lindquist generates both negative and positive feedback. We can see this from the follow phrase: But,  $40 * m$  is correct for the distance rowed so far. If the system just blatantly spits out a "That is incorrect!" or some other admonishing phrase, the student might believe that the entire answer was completely incorrect. But with the help of positive feedback generation, Ms. Lindquist can notify the student that one or more parts of the answer are actually correct. This shifts the student's focus to the problematic portion(s) of the answer.

The dialogue shown above illustrates dynamic scaffolding without strategies. Ms. Lindquist simply breaks the problem down into several parts, in this case  $800 - 40 * m$  is broken down into 800 and  $40 * m$ , and asks about the part that the student answered incorrectly. Although this is functional, it's not very educational; the dialogue generated would look very similar to the one above for every single problem, which has no embedded hints that would assist the student in getting the right answer.

This is why Ms. Lindquist uses strategies. Strategies are ordered sets of questions that represent ways of looking at the same problem from different angles. This not only adds variety to the problem-solving process, but also allows the ITS to provide helpful hints to the student when necessary. The following four sections describe these strategies in detail.

### 1.6.2 Strategy 1: Concrete Articulation

Ms. Lindquist's algebraic expressions always contain variables. For example, a problem would never have the final answer  $8 * (5 + 16)$ , but rather something like  $8 * (x + 16)$ . Otherwise, students would be tutored not on how to think algebraically when solving such expressions, but rather on simple numerical computations. That's not Ms. Lindquist's goal. However, simple numerical computations can be used to guide the student to the correct answer. The usage of variables allows the ITS to plug values into the variable(s), which would be  $x$  in the previous example, and ask the student what the overall numerical value of the expression is. A sample dialogue is shown below.

**Problem:** Anne is rowing a boat in a lake and is 800 yards from the dock from which she started. She rows back towards the dock at 40 yards per minute for  $m$  minutes and stops to rest. How far is she from the dock now?

**Tutor:** Hello. Please write an expression for the distance Anne has left to row.

**Student:**  $800 + 40 * m$

**Tutor:** No. But,  $40 * m$  is correct for the distance rowed so far. We will come back to the distance Anne has left to row. Let me try to break this down for you. Please compute the distance Anne has left to row if the number of minutes rowing was 3 minutes?

**Student:** 680

**Tutor:** Correct. Using only 800, 40, and 3 write an expression for the distance Anne has left to row.

**Student:**  $800 - 40 * 3$

**Tutor:** Correct. Now use  $m$  instead of 3 to again express the distance Anne has left to row.

**Student:**  $800 - 40 * m$

**Tutor:** Correct. You solved the entire problem.

Formulating an algebraic expression from a problem description is a problem for many students; therefore, this strategy tests if the student has a general understanding of the problem by asking for the solution for a concrete instance. First, Ms. Lindquist asks the student to calculate the numerical value of answer by substituting 3 for  $m$ . Second, the ITS asks the student to write the expression  $800 - 40 * 3$  to reflect how the numerical value was calculated. Finally, the student is instructed to replace the 3 with the variable  $m$  in order to get the final answer. The concrete articulation strategy utilizes these three main questions to tutor the student.

### 1.6.3 Strategy 2: Explain in English First

Ms. Lindquist can also assist the student in finding the correct answer by specifically asking how different parts of the problem are related to each other. The strategy temporarily puts mathematical concepts aside and focuses on assembling the answer from descriptions found in the problem statement. The student is required to piece together three English phrases using a set of pull-down menus to come up with the answer; this is demonstrated in the following dialogue:

**Problem:** Anne is rowing a boat in a lake and is 800 yards from the dock from which she started. She rows back towards the dock at 40 yards per minute for  $m$  minutes and stops to rest. How far is she from the dock now?

**Tutor:** Hello. Please write an expression for the distance Anne has left to row.

**Student:**  $800 + 40 * m$

**Tutor:** No. But,  $40 * m$  is correct for the distance rowed so far. We will come back to the distance Anne has left to row.

Let me try to break this down for you. Can you explain how you would find the distance Anne has left to row?

**Student:** the distance Anne has left to row is equal to the distance she started from the dock minus her speed rowing.

**Tutor:** Hmm. No. Figure out some way of using both the distance she started from the dock and the distance rowed so far.

**Student:** the distance Anne has left to row is equal to the distance she started from the dock minus the distance rowed so far.

**Tutor:** Correct. Now, follow that up by writing an expression for the distance Anne has left to row.

**Student:**  $800 - 40 * m$

**Tutor:** Correct. You solved the entire problem.

As we can see, this strategy looks at the problem from a completely different angle than the concrete articulation strategy. It only uses two main questions for tutoring. Ms. Lindquist first instructs the student to piece together a description in English, using a set of pull-down menus, of what the expression should look like. This can be seen from the second and third answers in the dialogue above. The separately underlined portions represent selections from one of the three pull-down menus. The answer “the distance Anne has left to row is equal to the distance she started from the dock minus her speed rowing” has separate pull-down menus for “the distance she started from the dock,” “minus,” and “her speed rowing.” Once the correct sequence of phrases has been chosen, Ms. Lindquist asks the student to convert the phrases into their corresponding arithmetic expressions. In this case, it’s  $800 - 40 * m$ , because “the distance she started from the dock” is 800 and “the distance rowed so far” is  $40 * m$ . As a result, the final answer is found.

### 1.6.4 Strategy 3: Convert the Problem into an Example to Explain

This strategy starts out by telling the student the correct answer, allowing the ITS to break down the solution into smaller portions and asking the student what each of the portions represents. The methodology is similar to the one above; phrases from pull-down menus are used to assemble a description of the expression. This strategy, however, asks the English meanings of different parts of the expression as opposed to instructing the student to convert the set of phrases into an expression. The following dialogue illustrates this method in action:

**Problem:** In math class there are 12 more boys than girls. If the number of girls is  $g$ , write an expression for the total number of students in the class.

**Tutor:** Hello. Please write an expression for the total number of students in a class.

**Student:**  $2 * g - 12$

**Tutor:** Hmm. No. The answer is actually  $g + (g + 12)$ . Let me ask you some questions to help you understand it. Can you explain where the  $g + 12$  came from?

**Student:**  $g + 12$  is equal to the total number of students in the class minus the number of girls.

**Tutor:** Hmm. No. But, that is only because I was looking for something else. What you said was true, but I was looking for the following: The first menu should be the number of girls. The second menu should be plus. And the third menu should be the difference between the number of boys and girls.

**Student:**  $g + 12$  is equal to the number of girls plus the difference between the number of boys and girls.

**Tutor:** Correct. What does  $g + 12$  represent?

**Student:** the number of boys

**Tutor:** Correct. Can you explain where the  $g + (g + 12)$  came from?

**Student:**  $g + (g + 12)$  is equal to the number of boys plus the number of girls.

**Tutor:** Correct. What does  $g + (g + 12)$  represent?

**Student:** the total number of students

**Tutor:** Correct. You solved the entire problem.

As in the previous section, the underlined portions of the student answers represent a single selection from a pull-down menu. The number of questions Ms. Lindquist will generate depends on the complexity of the mathematical expression. In this case, the system only asks for the English descriptions of  $g + 12$  and  $g + (g + 12)$ . Unlike the previous strategy, this one uses a combination of one or three pull-down menus to allow the student to construct the description. This can be observed from the two different answers “the total number of students” and “ $g + (g + 12)$  is equal to the number of boys plus the number of girls” respectively. This is also the only strategy that starts out with the final answer and works backwards to ensure that the student fully understands it.

### 1.6.5 Strategy 4: Introduce a New Variable

In order to help the student visualize the problem, Ms. Lindquist uses this strategy to group together portions of the answer into a single variable; this simplifies the task of constructing the correct expression. It first asks the student to find a relationship between different portions of the problem using the newly introduced variable(s), and only then does it ask for the final expression. The following demonstrates this process:

**Problem:** Julie was trying to raise money to help fight cancer. She got 7 students to each donate  $s$  dollars and  $t$  teachers to each donate 10 dollars. Write an expression for how much she collected?

**Tutor:** Hello. Please write an expression for the total amount Julie collected.

**Student:**  $7 * s - t * 10$

**Tutor:** No. But,  $7 * s$  is correct for the total amount the students donated, and  $t * 10$  is correct for the total amount the teachers donated. We will come back to the total amount Julie collected. Let me try to break this down for you. If  $C$  was the total amount the students donated and  $B$  was the total amount the teachers donated then what would be the total amount Julie collected?

**Student:**  $C + B$

**Tutor:** Correct. Now, let's try this question again: You are correct that  $C + B$  is the total amount Julie collected. Now, can you state this in terms of the numbers given in the problem?

**Student:**  $7 * s + t * 10$

**Tutor:** Correct. You solved the entire problem.

Ms. Lindquist takes  $7 * s$  and replaces it with the variable  $C$ , and then replaces  $t * 10$  with  $B$ . This allows the student to focus on the relationship between the two portions instead of the entire expression. This strategy is especially helpful with more complex expressions, such as  $5 * g + 7 * (30 - g)$ . If the student got the addition portion wrong, Ms. Lindquist would replace  $5 * g$  with  $A$  and  $7 * (30 - g)$  with  $B$ , and simply ask for  $A + B$ , which is easier to visualize. To summarize, this strategy is made up of two main questions; one of them introduces the variable(s) to simplify the problem, and the other asks the student to replace the variable(s) with the corresponding sub-expression(s).

## **1.7 Chapter Conclusions**

This chapter introduced some of the more sophisticated authoring tools available today. Authoring tools are becoming more and more domain-independent, allowing users to customize even the graphical user interface of their ITSs. However, neither of these tools would be an ideal choice for implementing a sophisticated ITS such as Ms. Lindquist. For example, Authorware is mainly an event-driven authoring tool that is able to produce visually impressive, interactive slideshows. This would not be sufficient for reproducing Ms. Lindquist, because there is no way to reconstruct the ITS's four teaching strategies. The chapter also elaborated on the most important architectural components of Ms. Lindquist; Mason needs to be able to emulate these components effectively. They are:



- 1.) A user interface that receives text input from the student and can display the feedback generated by the system.
- 2.) A diagnostic process that can single out the incorrect portions of the student's answer from the correct ones.
- 3.) Support for multi-step strategies.
- 4.) Ability to generate coherent dialog.

The goal of this thesis is to construct an authoring tool that not only makes ITS construction easier, but significantly shortens the development time. Mason can in fact be used to build a copy of Ms. Lindquist that supports all of the features listed above. This will be elaborated upon in the following chapters.

## Chapter 2: Mason's Architecture

In order for an authoring tool to be able to produce powerful Intelligent Tutoring Systems, it needs to provide support for the customization for the four core ITS components: the domain model, the student model, the teaching strategies, and the learning environment. Thus, one of the major decisions that had to be made during the implementation of the authoring tool was how much customizability would be supported for each ITS component.

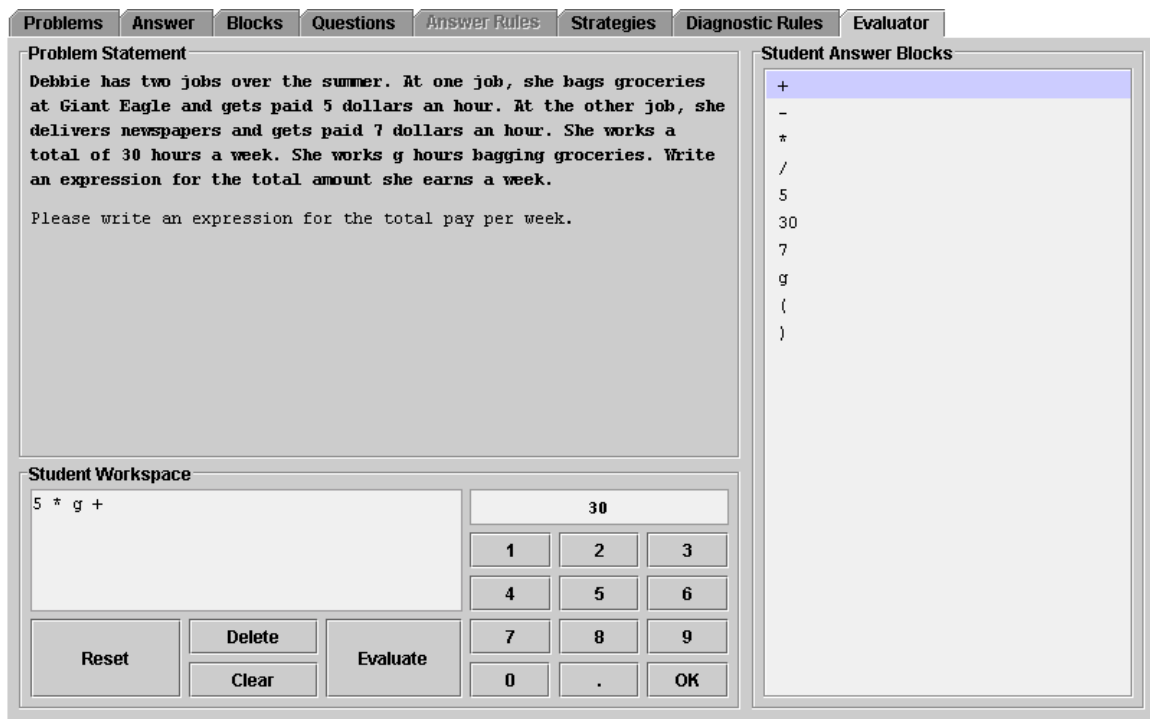


Figure 2 - The graphical user interface of the Evaluator tab

First, let's take a look at the learning environment. As specified previously, ITSs constructed with Mason have an interface similar to that of a chat room; the system outputs generated dialogue for the student, which consists of questions, feedback, and hints, and the student inputs the answer by constructing it in a small workspace. All answers are assembled from a provided set of answer blocks, which are dynamically

generated for the student. For example, if the tutoring system asks for the answer  $5 * g + 7 * (30 - g)$ , then it would make the following answer blocks available to the student: +, -, \*, /, g, (, and ). The answer blocks for the numerical values, 5, 7, and 30, need to be created by the student using the small calculator-like interface provided by Mason (see Figure 2). This makes the assembly of the answer less of a multiple choice. As a result, for  $5 * g + 7 * (30 - g)$ , the following would be an accepted answer:



The image shows a sequence of ten rectangular blocks arranged horizontally. From left to right, the blocks contain: the number '5', the asterisk symbol '\*', the letter 'g', the plus sign '+', the number '7', the asterisk symbol '\*', the left parenthesis '(', the number '30', the minus sign '-', the letter 'g', and the right parenthesis ')'. Each block has a thin black border and a slight drop shadow, giving it a 3D appearance.

**Figure 3 - An assembled student answer in Mason**

This method eliminates random “typos”, as any answer with an incomprehensible ordering of blocks will be marked as wrong and analyzed by the diagnostic engine. This is the only learning environment supported by the current version of Mason. Some of today’s commercial authoring tools, such as Authorware, can assemble user interfaces from images, sounds, and interactive components. Building an authoring tool that can produce ITSs with visually appealing graphical user interfaces is not the goal of this thesis. Thus, the learning environment isn’t customizable at all in Mason.

Next, let’s examine the student model. This is another ITS component that cannot be modified in Mason. The default functionality of the authoring tool’s student model is to simply keep track of which problem the student is currently working on. Some sophisticated ITSs store more information about the student. Ms. Lindquist, for example, keeps track of the types of problems the student answered erroneously and asks problems of similar nature for more effective tutoring. Mason doesn’t support the creation of such student models, because the thesis focuses on problem construction and dialogue generation rather than student tracking.

The remaining two components are what Mason excels at in customizing. In Mason, the teaching strategies component is merged with the domain model, since the strategies are simply ordered sets of question templates. The authoring tool's hierarchical architecture has several limitations. For example, for algebra, one of the limitations is the fact that Mason can only tutor the student using the infix notation; neither the prefix nor the postfix notations are supported. Currently, it's best to use the system for mathematical problems, although future versions should be more domain-independent and support tutoring for programming languages and other areas of study.

First, this chapter focuses on Mason's diagnostic system. This should provide a good overview of how elements from the domain model collaborate with the diagnostic methods in order to generate effective dialogue for the student. The fact that the complexity of the domain model is proportional to the effectiveness of the dialogue generated is also discussed. Then, this chapter describes the domain model and its elements in detail. The domain model is where most of the customizability lies.

## ***2.1 The Diagnostic System***

Without the diagnostic engine, there would be no dialogue generation. The engine is used to compare the student's answer to the desired one built by the author. If there is a slight or complete mismatch, then feedback is produced in order to focus on the erroneous parts of the student's answer. Let's take a look at the following dialogue, which was generated by Mason's replica of Ms. Lindquist:

**Problem:** Anne is rowing a boat in a lake and is 800 yards from the dock from which she started. She rows back towards the dock at 40 yards per minute for  $m$  minutes and stops to rest. How far is she from the dock now?

**Tutor:** Please write an expression for the distance Anne has left to row.

**Student:**  $800 + 40 * m$

**Tutor:** No. But,  $40 * m$  is correct for the distance rowed so far. We will come back to the distance Anne has left to row. Let me try to break this down for you. Can you explain how you would find the distance Anne has left to row?

**Student:** the distance Anne has left to row is equal to the distance she started from the dock minus her speed rowing.

**Tutor:** Hmm. No. Figure out some way of using both the distance she started from the dock and the distance rowed so far.

**Student:** the distance Anne has left to row is equal to the distance she started from the dock minus the distance rowed so far.

**Tutor:** Correct. Now, follow that up by writing an expression for the distance Anne has left to row.

**Student:**  $800 - 40 * m$

**Tutor:** Correct. You solved the entire problem.

In the dialogue, Mason uses the *Explain in English First* strategy after the student inputs the incorrect expression. How does Mason know exactly which strategy to use? In order to answer this question, we need to take a look at the system's architecture. In the domain models of ITSs constructed with Mason, every single problem has its corresponding answer structure.

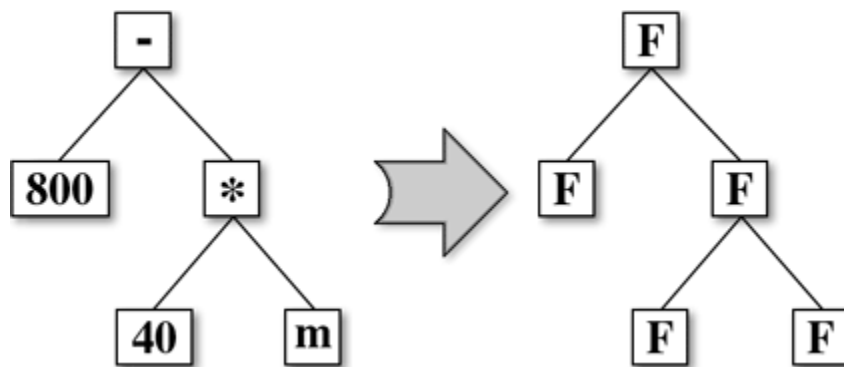


Figure 4 - An answer tree mapped onto a boolean tree

The answers are represented hierarchically as a tree, where each node represents an answer portion that is also known as a **problem partition**. For example, the equation

$800 - 40 * m$  would have four different nodes in its tree representation: `-`, `*`, `800`, `40`, `m`.

Hence, the domain model is hierarchical. Each answer tree can be mapped onto a **boolean tree** with an identical node structure; the boolean values represent the correctness of their corresponding problem partitions in the answer tree (see Figure 4).

Mason uses a bottom-up approach to diagnose student answers. The purpose of the diagnostic system is to (i) find the lowest faulty node in the answer tree if the student answered the problem incorrectly, (ii) use the provided set of diagnostic rules to determine which set of strategies is applicable to the faulty node, and (iii) pass the faulty node and a selected strategy onto the dialogue generation system. A **faulty node** in the answer tree is defined as a node whose correspondent in the boolean tree is set to `false`. Since each node in the answer tree represents a problem partition, Mason can pinpoint parts of the problem the student had difficulty with just by searching for faulty blocks. Hence, the hierarchical representation is very effective for the diagnostic process, which involves the following steps:

- 1.) Before the student inputs the very first answer to the problem statement, all of the boolean tree nodes are set to `false`. This, by default, indicates that no problem partitions have been answered correctly.
- 2.) All of the possible answer combinations are generated from the root node of the answer tree (taking the commutativity of applicable nodes into account), and are compared to the answer assembled by the student. If a match is found, then the entire problem is considered

solved; the remaining steps of the diagnostic process can be ignored.

For  $800 - 40 * m$ , the following would be marked as correct:

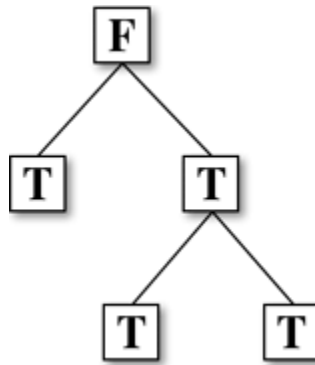
$$\begin{array}{l} 800 - 40 * m \\ 800 - m * 40 \end{array}$$

If none of the combinations match the student's answer, then a more thorough analysis is necessary to find faulty nodes. Mason moves onto the next diagnostic step.

- 3.) All of the possible combinations are generated for every single node of the answer tree. In other words, all the possible problem partitions are stored in a list. The student's answer is then scanned for the existence of each combination. If one of the combinations for a particular node is found in the answer, then the node's correspondent in the boolean tree is set to `true`. This signifies that the student answered that partition correctly. Otherwise, the node will stay `false`. For  $800 - 40 * m$ , the student's answer would be scanned for each of the following:

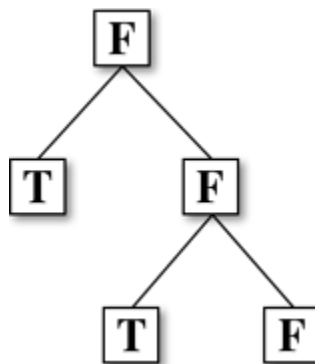
$$\begin{array}{l} 800 \\ 40 \\ m \\ 40 * m \\ m * 40 \\ 800 - 40 * m \\ 800 - m * 40 \end{array}$$

By the end of the process, the boolean tree will be filled with `true` and `false` values, indicating exactly which parts of the answer the student answered incorrectly. This allows Mason to generate the appropriate dialogue only for problem partitions that the student needs to be tutored on. For example, if the student answered  $800 + 40 * m$  instead of  $800 - 40 * m$ , the boolean tree would look as follows:



**Figure 5 - A boolean tree where only the root is a faulty node**

This indicates that the student did not use the  $-$  operator where it was necessary.  $800$ ,  $40$ ,  $m$ , and  $40 * m$  all exist in the student's answer, but  $800 - 40 * m$  does not. Of course, more serious errors can be made as well. For example, if  $800 - 40$  is input as the answer, then the boolean tree will have different boolean values:



**Figure 6 - A boolean tree with three faulty nodes**

After all, only  $800$  and  $40$  show up in the answer, and  $m$ ,  $40 * m$  (or  $m * 40$ ), and  $800 - 40 * m$  (or  $800 - m * 40$ ) do not.

- 4.) Search the boolean tree for the deepest faulty node. The corresponding node in the answer tree is the deepest problem partition the student needs to be tutored on. In the figure above, the deepest faulty node



would be the one for the variable  $m$ , because  $m$  was not found in the input answer.

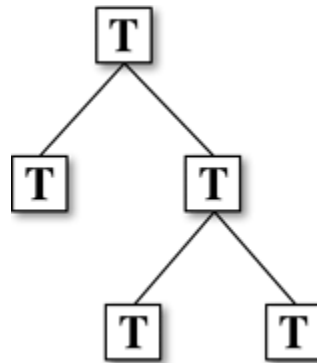
- 5.) Using a provided set of diagnostic rules, determine which set of strategies is applicable for the faulty node. For example, the author may have a diagnostic rule constructed for detecting missing parentheses. Remember that tutoring strategies are used for generating the appropriate pedagogical dialogue for the student regarding the problematic problem partition. Thus, diagnostic rules can be used to determine what the problem might be with the student's answer, and then pick the suitable tutoring strategy for it.
- 6.) Finally, pass the faulty node and the selected tutorial strategy onto the dialogue generation system.

When the student's very first answer to the problem statement is being diagnosed, the next faulty node is always the deepest one. Once again, in Figure 6, this node would be the one for the variable  $m$ . Afterwards, Mason switches to a bottom-up approach for tackling the problem partitions. The system will keep on asking questions (or a set of questions) about each faulty node in the tree until the root node is reached. The root node, of course, would be the start of the problem. This short example should clarify this process:

- 1.) Mason displays the problem statement and asks for the required expression. The answer Mason is looking for is  $800 - 40 * m$ .
- 2.) The student inputs  $800 - 40$ .

- 3.) Mason generates negative feedback, and, as shown in Figure 6, finds the node for the variable  $m$  to be the deepest faulty node. Using a set of diagnostic rules, it picks an appropriate strategy and passes the node and the strategy onto the dialogue generation system.
- 4.) After one or more questions generated by the domain model, the student finally understands the purpose of the variable  $m$ ; therefore, the boolean correspondent of the node for the variable  $m$  is set to `true`.
- 5.) Mason now searches for the next faulty node, which ends up being the node for the  $*$  operator. Of course, the node represents the problem partition  $40 * m$ . This is also passed along for dialogue generation.
- 6.) Once all the necessary questions are asked about  $40 * m$  and the student understands its purpose, the boolean correspondent of the  $*$  node is marked `true` as well.
- 7.) Mason searches for the next faulty node in the answer tree, and this time it ends up being the root node that represents the  $-$  operator. The system is back at the top of the tree, which means that all the student needs to do now is understand  $800 - 40 * m$ . Thus, once again, dialogue is generated for the root node.
- 8.) Once the student successfully assembles the final answer to the problem, the boolean value for the root node will be set to `true` as well. If the student gets the equation wrong again, Mason is not going to restart the tutoring process. The system will never choose a child of

the current node as the next faulty node. When the final question is answered correctly, the boolean tree will have the following structure:



**Figure 7 - A boolean tree indicating that the problem has been solved**

Only when all the nodes are set to `true` is the problem considered to be fully solved. This state of correctness indicates that the student understands all of the problem partitions.

Mason also supports a top-down method for generating positive feedback. Unlike the bottom-up search for faulty blocks, the generation of positive feedback involves finding the largest problem partitions that the student has answered correctly. For example, when the student inputs  $800 + 40 * g$  as the answer instead of  $800 - 40 * g$ , Mason compliments the student for getting the  $40 * m$  partition correct.

In a nutshell, the purpose of the diagnostic engine is to find all the errors in the student's answer (if there are any), provide very basic positive and negative feedback, and pass the erroneous problem partitions along with a tutorial strategy onto the domain model for advanced dialogue generation. The following section is dedicated entirely to the domain model.

## 2.2 The Domain Model

To best understand the domain model's architecture, let's break it down into five main components: **problems**, **questions**, **strategies**, **strategy sets**, and **diagnostic rules**. Problems, or exercises, consist of a problem statement and its corresponding answer, which is represented hierarchically as a tree. Blocks are used to construct the tree representations of the problem answers; these trees were introduced in first section of this chapter. Questions are templates for determining the generated dialogue for the student; they add flexibility to the system, since a single, well-designed template can be applied to many different problems. As mentioned previously, a strategy is simply a way of asking an ordered set of questions. The sample dialogue below, generated for a problem which we will refer to as the "Debbie" problem from now on, demonstrates a strategy in action.

**Problem:** Debbie has two jobs over the summer. At one job, she bags groceries at Giant Eagle and gets paid 5 dollars an hour. At the other job she delivers newspapers and gets paid 7 dollars an hour. She works a total of 30 hours a week, and she works  $g$  hours bagging groceries. Write an expression for the total amount she earns a week.

**Tutor:** Please write an expression for the total pay per week.

**Student:**  $30 * ( 7 * 5 )$

**Tutor:** Hmm. No. We will come back to the total pay per week. Let me try to break this down for you. What is the number of hours for delivering newspapers?

**Student:**  $30 - g$

**Tutor:** Correct. If  $B$  was the hours delivering newspapers then what would be the pay from the newspaper?

**Student:**  $B * 7$

**Tutor:** Correct. You are correct that  $7*B$  is the pay from the newspaper. Now, can you state this in terms of the numbers given in the problem?

**Student:**  $7 * b + 5 * g$

**Tutor:** Hmm. No. What is the pay from the newspaper? Hint: You know the answer is  $7 * B$  but you must substitute in the value for  $B$ . Replace  $B$  with  $30 - g$ . Use parentheses if needed.

**Student:**  $(30 - g) * 7$

**Tutor:** Correct. What is the pay from the grocery?

**Student:**  $(30 - b) * 5$

**Tutor:** Hmm. No. Use 5 and g to calculate the pay from the grocery.

**Student:**  $5 * g$

**Tutor:** Correct. If D was the pay from the grocery and C was the pay from the newspaper then what would be the total pay per week?

**Student:**  $D + C$

**Tutor:** Correct. Now, let's try this question again: You are correct that  $D + C$  is the total pay per week. Now, can you state this in terms of the numbers given in the problem?

**Student:**  $(30 - g) * 5 + (30 - b) * 7$

**Tutor:** Hmm. No. What is the total pay per week? Hint: You know the answer is  $D + C$  but you must substitute in the values for D and C. Replace D with  $5 * g$  and replace C with  $7 * (30 - g)$ . Use parentheses if needed.

**Student:**  $(5 * g) / (7 * b)$

**Tutor:** Hmm. No. Please input:  $5 * g + 7 * (30 - g)$ .

**Student:**  $5 * g + 7 * (30 - g)$

**Tutor:** Correct.

This dialogue was generated using a complex domain model structure and the diagnostic engine described in the previous section. It demonstrates the *Introduce a New Variable* strategy in use. As we can observe, it's very similar to the dialogue Ms. Lindquist can produce. This is due to the fact that the strategies built in Mason can emulate Ms. Lindquist's strategies well.

Strategy sets are Mason's solution for randomizing tutorial strategies. Randomization allows for more varied tutoring sessions; it would be repetitive to use the same strategy over and over again for each error the student makes. Diagnostic rules determine what strategy sets get used for a particular error in the student's answer. Well-constructed rules can not only reduce ITS development time, but also make the updating process of the system easier. For example, a rule can be written for determining that a

student is missing a required pair of parentheses, and it can be bound to a strategy that asks questions about the missing parentheses only. This section describes each of the domain model components in detail, and demonstrates how they are used to generate the dialogue for the “Debbie” problem.

### 2.2.1 Problems, Blocks, and Answer Trees

When Mason is launched by the author, its first screen, the *Problems* tab, displays the problem listing (see Figure 8). Problems are nothing more than pairs of problem statements and their corresponding answers; the answers are represented as trees. They are also fully independent from questions, strategies, strategy sets, and rules. In order to construct a problem, it’s best to start with the **problem statement**. If the problem statement is defined early on in the construction process, it can aid the author in designing the answer tree and its corresponding elements.

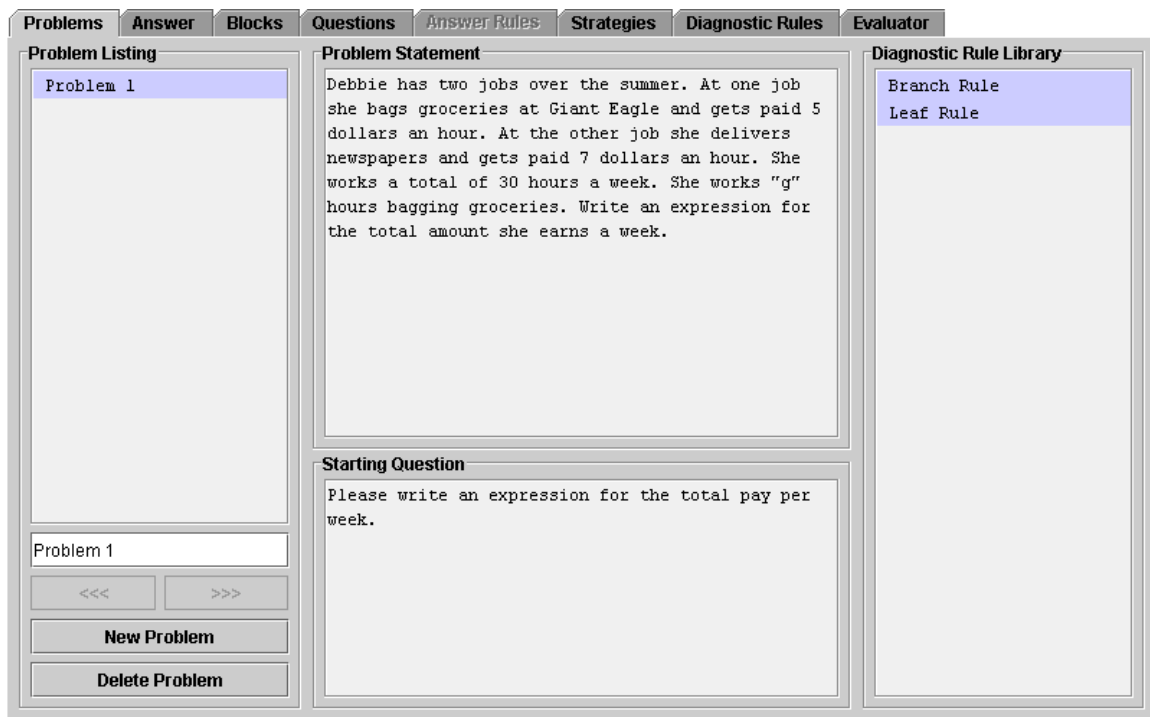


Figure 8 - The graphical user interface of the Problems tab

For example, how complex will the answer be for the problem? Roughly how many blocks will the answer tree require? Spelling out the problem statement in the beginning can answer numerous design questions early in the problem construction process. In case of the “Debbie” example shown above, the problem statement is:

**Problem:** Debbie has two jobs over the summer. At one job, she bags groceries at Giant Eagle and gets paid 5 dollars an hour. At the other job she delivers newspapers and gets paid 7 dollars an hour. She works a total of 30 hours a week, and she works  $g$  hours bagging groceries. Write an expression for the total amount she earns a week.

Mason’s user interface automatically updates the system as the user enters text into the text box. This implementation is a minor user interface enhancement that reduces the number of necessary mouse clicks. Once the problem statement is typed in, an **initial question** needs to be provided; this is the very first question displayed for the student regarding the problem. In the example dialogue, “Please write an expression for the total pay per week.” was used.

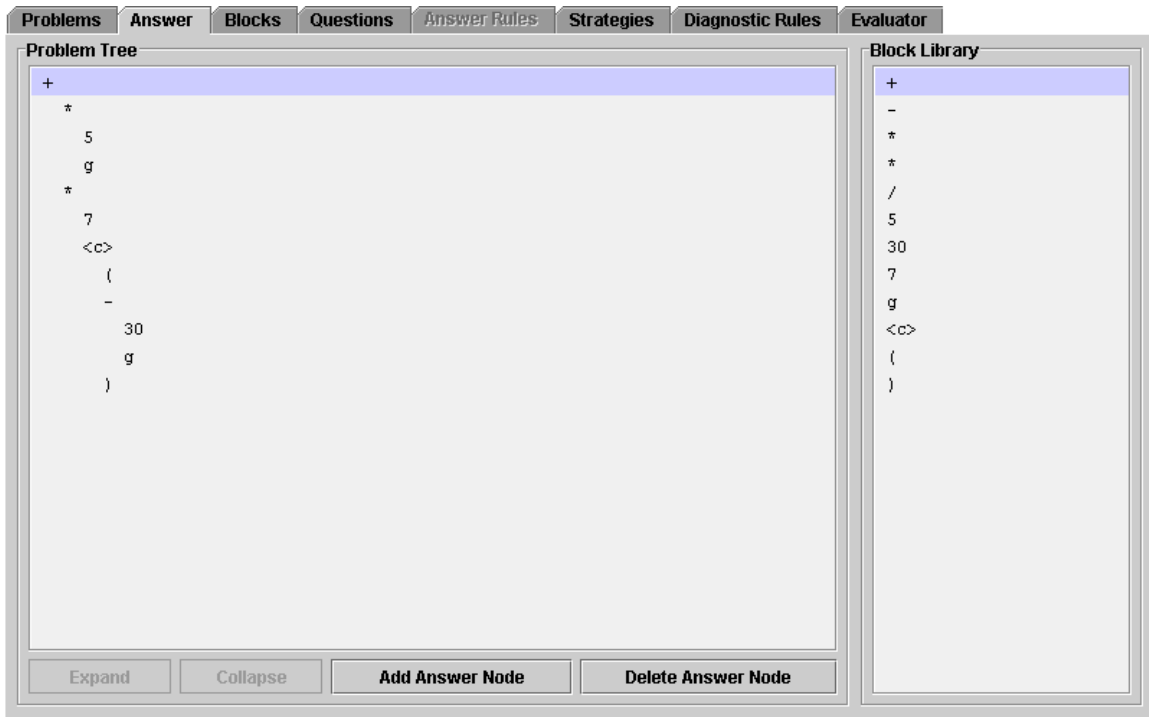


Figure 9 - The graphical user interface of the Answer tab

The *Problems* tab also displays the diagnostic rules available for usage. The rule selection feature is discussed later on in this chapter. By now, the user should have a clear concept of what the answer tree should look like. This brings us to the *Answer* tab, which displays the entire tree structure. The user can construct the entire structure, which starts out with an empty root node for new problems, in order to get a clear idea of what blocks will be needed. As an example, let's take a look at the answer to the "Debbie" problem. It's clear that the desired expression for the problem is:

$$5 * g + 7 * (30 - g)$$

This answer can be simplified or expanded further, of course, but all of the different answer variations can be deduced from the expression above. Like all simple algebraic expressions, it can be represented as a tree (see Figure 10). Note that Mason currently supports the infix notation only. The node depicted as a circle is a **connector block**, which is simply used to concatenate a set of blocks together in a specific order. Its functionally is explained later in this section.

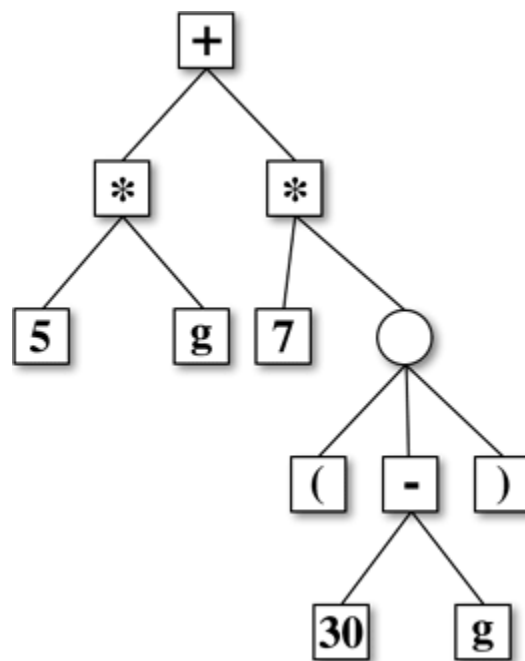


Figure 10 - The tree representation of an expression



The tree needs to be constructed under Mason's *Answer* tab. The following illustrates the tree structure without any blocks attached to it:

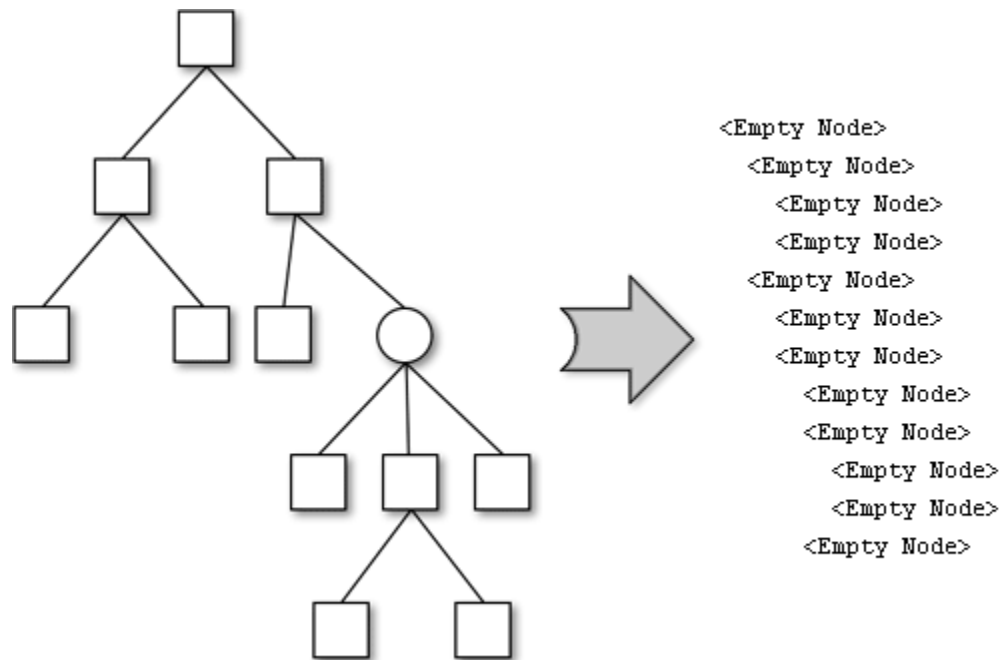


Figure 11 - The tree structure of the equation with empty nodes

If there are no blocks bound to the tree, Mason will simply display the structure with empty nodes (see Figure 11). The desired expression for the “Debbie” problem consists of nine elements; these are: 5, 7, 30, g, +, -, \*, ( and ). Thus, the user should be able to complete the answer tree of the problem by creating nine different blocks and bind them to the tree structure. Nine elements, however, might not be sufficient. In order to find the exact number needed, it’s necessary to understand the purpose of blocks.

In Mason, **blocks** are the building elements of problems; they are bound to the nodes of answer trees. A block comes with a set of fully customizable attributes. Additional attributes can be added by the tutor if necessary; this is often the case especially when constructing complex question templates. For example, let’s take a look

at the block for the number 5 in the answer to the “Debbie” problem. It would have the following attribute slots:

|          |                     |                                       |
|----------|---------------------|---------------------------------------|
| <b>5</b> | <b>Symbol:</b>      | 5                                     |
|          | <b>Type:</b>        | number                                |
|          | <b>Description:</b> | her hourly wage for bagging groceries |
|          | <b>Ignore:</b>      | false                                 |
|          | <b>Commutative:</b> | false                                 |

These attributes, also referred to as **system attribute slots**, are available for all newly created blocks. They cannot be removed, and their labels, such as `Symbol` and `Type`, cannot be modified. Their values, however, are essential not only for the diagnostic engine, but also for dialogue generation. The first of these slots, `Symbol`, is the block’s visual representation in Mason’s graphical user interface. Accordingly, the number 5 would be represented with a block with the character 5 for its `Symbol`. The number 53 would not be pieced together from two different blocks; instead, it would be represented with a single block with 53 for its `Symbol`.

Mason knows nothing about the mathematical meaning of the attribute values; the system cannot identify the block 53 as an object with an actual numerical value. The attribute slot values are stored as strings and booleans, and are simply used for diagnosis and generation. The user needs to keep this in mind while developing an ITS with Mason. Although this representation has its drawbacks, it is effective for constructing complex ITSs; the advantages and disadvantages are elaborated upon in the following chapter during the discussion of the evaluation. Let’s ignore the other system attributes for a moment, and finish the construction of the answer tree for the Debbie problem. A block can be bound to one or more tree nodes right after its `Symbol` attribute has been specified:

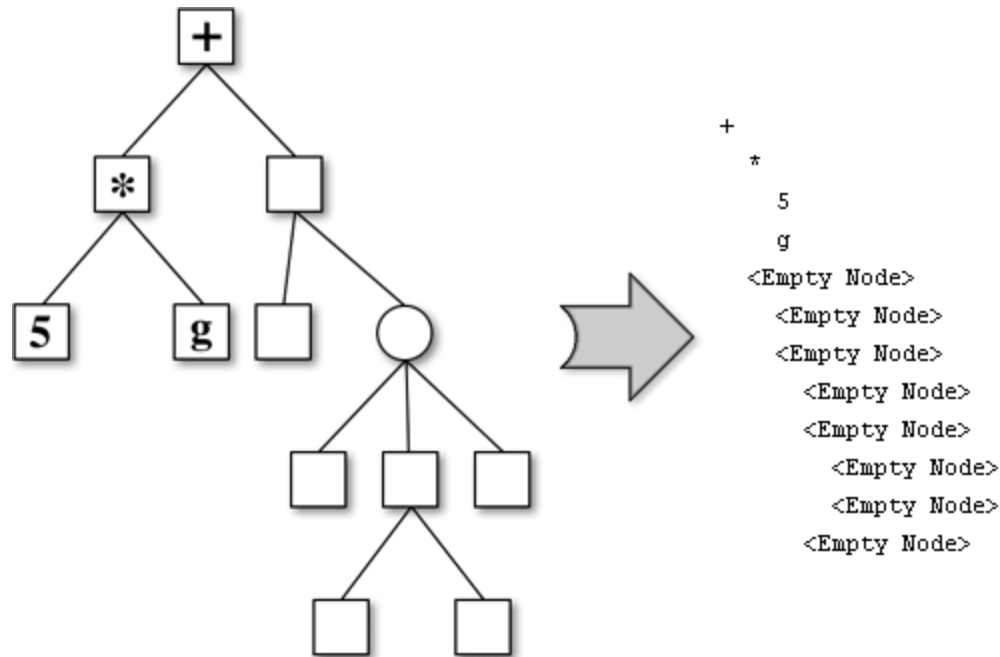


Figure 12 - Binding blocks to the answer tree

Thus, once the blocks are created for 5, 7, 30, g, +, -, \*, ( and ), which can be done under the *Blocks* tab, and are bound to their corresponding answer tree nodes under the *Answer* tab, the final the answer tree can be completed (see Figure 13). The actual process of binding blocks to nodes is rather simple. All the author needs to do is select a node in the tree, and then select its corresponding block. Mason automatically updates the tree accordingly. This concludes the construction of a problem and its answer. Mason is unable to tutor the student given such a limited amount of information, so let's move on.

The remaining system attributes, *Type* and *Description*, are used to help the user identify the block. For example, *number*, *operator*, *letter*, *car*, and *spaceship* are all strings that can be used to identify the block's *Type*. *Description* usually describes exactly what the block represents. In the "Debbie" problem, for example, 5 represents "her hourly wage for bagging groceries". Mason can already produce realistic dialogue simply by plugging in the values of *Type* and *Description* into templates.

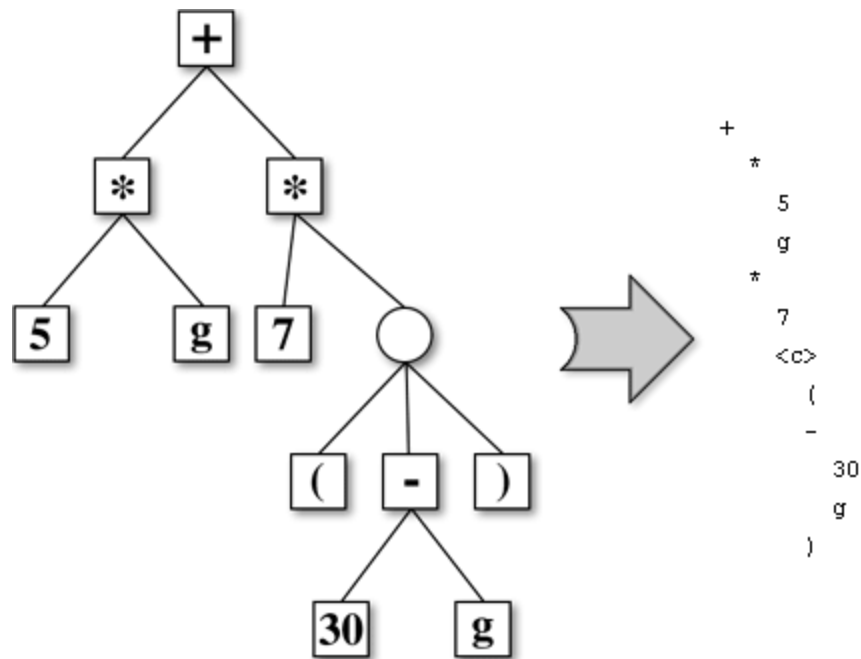


Figure 13 - The complete answer tree

The system attributes `Ignore` and `Commutative` have boolean values. In order for Mason to ignore a block during the diagnostic process of the student's answer, its `Ignore` attribute needs to be set to `true`. For example, the connector block (represented as a circle in the diagram above) should be ignored during diagnosis, since it's only used during the construction of the answer tree. Requiring the connector block in the student answer would simply cause confusion on the student's part.

The last system attribute slot, `Commutative`, defines whether or not the block and its children adhere to the commutative property. In other words, the order of a commutative node's children doesn't count. Recall that the problem's answer is structured as a tree. This is a powerful way of representation, because trees can be manipulated easily. For example, we can represent both  $5 * g$  and  $g * 5$  (due to the commutative property) simply by switching the order of the `*` node's children.

Consequently, this representation seems powerful for problem answers for which a hierarchical representation is ideal.

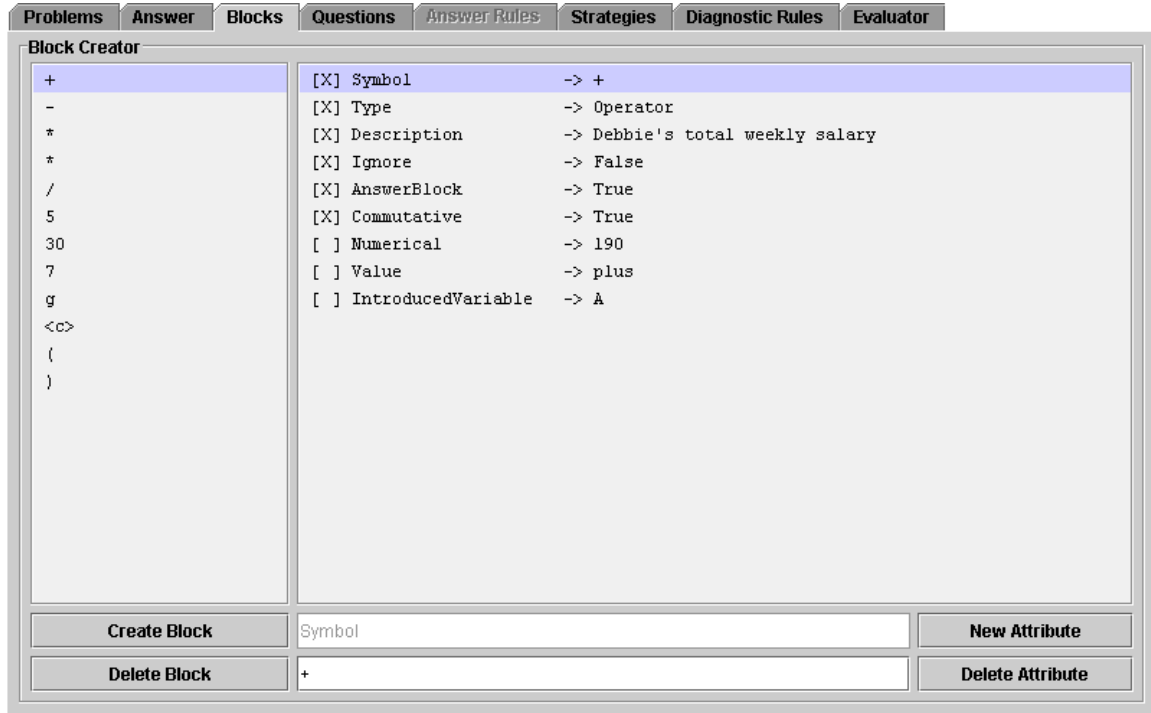
What if the tutor wanted to make a spelling tutoring system? A spelling problem would be very hard to represent in a similar hierarchical fashion. One way would be to break words up into sets of letters, but that can result in complications; however, an algebra problem, as the Debbie problem illustrates, can be easily represented in a tree. Also recall that the current version of Mason knows nothing about the meaning of the blocks; it doesn't understand that the block 5 has a numerical meaning. What Mason does know about is the structure of the answer tree and, consequently, the order of the blocks in that tree. This is exactly where the `Commutative` system attribute is utilized.

`Commutative` is an essential attribute, especially since the ordering of blocks plays such an important role in Mason. Without commutativity, the only accepted answer for the Debbie problem would be  $5 * g + 7 * (30 - g)$ ; the commutative property makes Mason's system more flexible, allowing it to accept all of the following answers:

$$\begin{aligned} &5 * g + 7 * (30 - g) \\ &g * 5 + 7 * (30 - g) \\ &5 * g + (30 - g) * 7 \\ &g * 5 + (30 - g) * 7 \\ &7 * (30 - g) + 5 * g \\ &(30 - g) * 7 + 5 * g \\ &7 * (30 - g) + g * 5 \\ &(30 - g) * 7 + g * 5 \end{aligned}$$

The more commutative blocks an answer tree has, the higher the number of possible answer combinations is. Of course, the user isn't limited to using the system attributes of blocks only. The author can create additional ones, called *dynamic attribute slots*, as well. Unlike system attributes, dynamic ones have modifiable labels, and can be removed from the system at any time. Let's take a look at a quick example for illustrating the use of dynamic attributes. Suppose we want to give the block `g` a numerical value.

Let's create a dynamic attribute called `Numerical` for it, and set its value to 10. Now, suppose that we want to specify a numerical value for the `*` block in the expression  $5 * g$  as well. Since we know that  $5 * g = 50$  when  $g = 10$ , we can create a `Numerical` attribute for the `*` block and set its value to 50. This flexibility adds a new dimension to dialogue generation.



**Figure 14 - The graphical user interface of the Blocks tab**

In Mason, the dynamic attribute slots are slightly distinguished from system attribute slots under the *Blocks* tab (see Figure 14). Dynamic slots have a `[ ]` in front of their labels, whereas system ones are indicated with a `[X]`. This concludes the section on problem representation and the building blocks of Mason. In the following section, the reader is introduced to exactly how block attributes are utilized by question templates for dialogue generation.

## 2.2.2 Questions, Templates, and Answer Rules

In order to guide the student to the correct solution, pedagogical questions need to be generated for the problem partitions that prove to be difficult for the student. In Mason, all questions are generated from **question templates**, which can be constructed under the *Questions* tab (see Figure 15). Templates contain dynamic elements whose values change depending on which answer tree node the template is being applied to.

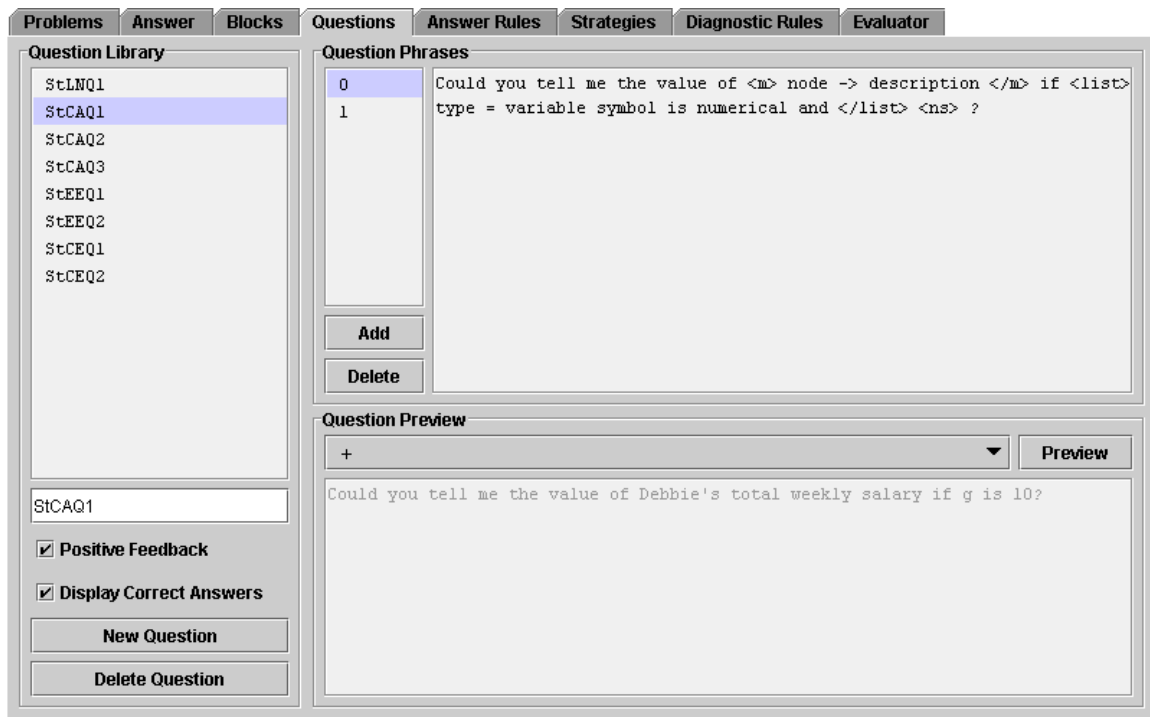


Figure 15 - The graphical user interface for the Question tab

These dynamic elements are specified using a set of keywords that are recognizable by Mason. Recall that the diagnostic engine determines the faulty node which the student is to be tutored on. When discussing question templates, the node passed onto the templates will be referred to as the *current node*. To illustrate how a template works, let's consider the following:

```
Please express <m> Node -> Description </m> using <m> Child
0 -> Description </m> and <m> Child 1 -> Description </m>
<ns> .
```

There are three dynamic elements in the phrase above; these are specified for the authoring tool using `<m>` tags (also known as Mason tags). Let's break the phrase down for easier understanding. The `<ns>` tag simply indicates that a space should not be displayed between the elements that directly precede and follow the tag. It is used for aesthetics only. The element `<m> Node -> Description </m>` is instructing Mason to retrieve the `Description` attribute of the block attached to the current node. For clarification, let's take a look what would be generated from the phrase if it is applied to the `*` node in the "Debbie" problem's `5 * g` expression. The following values are used for the `Description` attribute slot of each block:

|          |   |
|----------|---|
| <b>5</b> | <b>Symbol:</b> 5<br><b>Description:</b> her hourly wage for bagging groceries   |
| <b>*</b> | <b>Symbol:</b> *<br><b>Description:</b> her weekly salary for bagging groceries |
| <b>g</b> | <b>Symbol:</b> g<br><b>Description:</b> her hours spent bagging groceries       |

Thus, if the current node is `*`, then `<m> Node -> Description </m>` would result in the phrase "her weekly salary at Giant Eagle". Mason automatically recognizes that the author is trying to access the `Description` attribute of the current node. Of course, the current node isn't the only one whose block's attributes can be accessed. As demonstrated by the question phrase above, the children nodes can be accessed as well. `Child 0` for the `*` node would be the node for `5`. `Child 1` would be the one for `g`. Thus, if the entire question template were applied to the `*` node fired, the following would be generated:



Please express her weekly salary at Giant Eagle using her hourly wage and the hours she bags groceries.

Mason recognizes two more keywords: `parent` and `root`. These can be used the same way as the `node` keyword. The following examples should demonstrate this:

```
Please express <m> parent -> description </m> <ns> .  
Please express <m> root -> description </m> <ns> .
```

The `root` keyword simply references the root of the answer tree, while `parent` references the parent of the current node. Question templates have one more dynamic feature, which is the generation of dynamic lists. This is accomplished using `<list>`. Let's assume that we are trying to generate the following question for the root node of the "Debbie" answer tree:

```
What is Debbie's total weekly salary if g is 10?
```

We could hardcode this question into the system using the following template:

```
What is <m> node -> description </m> if g is 10?
```

This, however, is not a reusable solution. It will only work for problems with a single variable, namely `g`. The `<list>` structure was designed to solve this issue. Let's assume that the author wants to output all the variables under the current node in the answer tree for the student and assign numerical values to them. In order to achieve this, the author would need to create a non-reusable template for each problem if Mason did not support the dynamic generation of lists. In fact, a template would be required for each problem partition. This `<list>` functionality has the following syntax:

```
<list> [slot value] = [value] [slot value] [connector]  
[slot value] [coordinating conjunction] </list>
```

Since the syntax is confusing, let's try to break it down. The `[slot value] = [value]` portion of the phrase is the regular expression. An example for this would be `type = number`. If the regular expression returns true for any of the blocks in the answer

tree, then the block will be used for the listing. The [slot value] [connector] [slot value] portion defines what the printout will look like. When we are trying to print out “g is 10”, then Symbol would be the first slot value, “is” would be the connector, and Numerical would be the second slot value. Thus, the syntax would be “symbol is numerical”. Finally, the [coordinating conjunction] part is for long lists. This is usually either “and” or “or”. It’s best if we demonstrate this for our example above. The correct syntax for generating “What is Debbie’s total weekly salary if g is 10?” is the following:

```
What is <m> node -> description </m> if <list> type = variable  
symbol is numerical and </list> <ns> ?
```

What happens if the “Debbie” tree has another variable, h, with the Numerical value of 5? How about a third variable, m, with 20 for its Numerical slot? The system would print out the following phrases for the answer tree variations with the two and three variables respectively:

```
What is Debbie’s total weekly salary if g is 10 and h is 5?  
What is Debbie’s total weekly salary if g is 10, h is 5, and m is  
20?
```

As we can observe, [coordinating conjunction] comes into play when there are two or more elements in the list. Questions also support rephrasing. Mason allows the author to add several different versions of the original question template to the same question. The replica of Ms. Lindquist, which is discussed in detail in the following chapter, has about three different phrases per question. The main purpose of phrases is to provide additional hints and suggestions for the student. When the student answers a question incorrectly, Mason does not load a different one into memory. Instead, it moves onto the next phrase (if possible) of the same question. In other words, it rephrases the

question. The following are three possible phrases Mason might ask from the student about “Debbie’s total weekly salary”:

- 1.) Please express her weekly salary at Giant Eagle using her hourly wage and the hours she bags groceries.
- 2.) Remember that her hourly wage for bagging groceries is 5 and that her hours spent bagging groceries is g.
- 3.) Please input:  $5 * g$ .

As we can see, the second phrase provides a hint for the user, and the third one just displays the desired answer. Remember that Mason only moves onto the next phrase if and only if the student answers the question incorrectly. Thus, the author can have many additional phrases with hints.

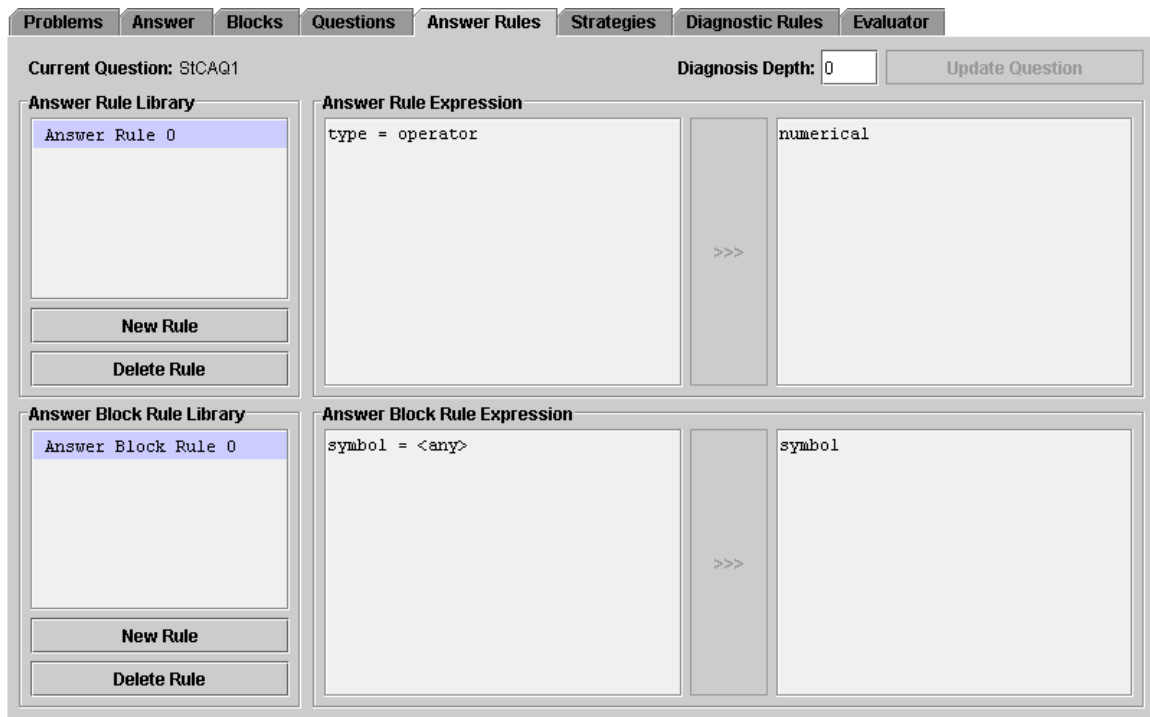


Figure 16 - The graphical user interface for the Answer Rules tab

Questions have two more important components: *answer rules* and *answer block rules*. These can be edited under the *Answer Rules* tab (see Figure 16). Answer rules play a key role in the diagnostic process. They determine which attribute slots of blocks the diagnostic system should compare to the student’s answer. After all, these comparisons

might differ from question to question. For example, one question might only be interested in comparing the student's answer to the blocks' `Symbol` values. Another one might want to use the `Value` attributes of operators, and the `Description` attributes of numbers. Although this concept might seem confusing at first, it makes the system very flexible; the same answer tree can be used for generating many different types of answers!

Let's take a look at another "Debbie" example for a demonstration, and assume that our blocks in the answer tree have the following slots:

|          |   |
|----------|---|
| <b>5</b> | <b>Symbol:</b> 5<br><b>Type:</b> number<br><b>Description:</b> her hourly wage for bagging groceries                            |
| <b>*</b> | <b>Symbol:</b> *<br><b>Type:</b> operator<br><b>Description:</b> her weekly salary for bagging groceries<br><b>Value:</b> times |
| <b>g</b> | <b>Symbol:</b> g<br><b>Type:</b> variable<br><b>Description:</b> her hours spent bagging groceries                              |

Once again, let's assume that the current faulty node is the left child of the + node in the answer tree, namely the \* node. Note how the `Value` attribute slot of the \* block is displayed in italic. It's a dynamic attribute slot added to that specific block only. Now, let's take a look at the following answer rule:

`symbol = <any>`    ➡    `symbol`

This simple rule, which is the default for both answer and answer block rules, indicates that for each block in the answer tree, the diagnostic system should only look at the `Symbol` attribute. The arrow represents a logical `if` statement, so the rule would translate to: if the `Symbol` attribute has any value, look at the `Symbol` attribute. The `<any>` keyword is reserved by Mason. Thus, for the three blocks above,

possible answers would be  $5 * g$  or  $g * 5$ , since we are only looking at the `Symbol` values. The rule could be used when, for example, the author wishes to ask for actual expressions from the answer tree. Now, let's take a look at the following answer rules:

|                              |   |                          |
|------------------------------|---|--------------------------|
| <code>type = operator</code> | ➡ | <code>value</code>       |
| <code>type = number</code>   | ➡ | <code>description</code> |
| <code>type = variable</code> | ➡ | <code>description</code> |

The above translates to the following:

- If the block is of `Type operator`, then use its `Value` attribute.
- If it's a number, then use its `Description` attribute.
- If it's a variable, then once again use its `Description`.
- Ignore all other blocks!

This rule set would require either of the following two answers from the student for the three blocks mentioned above:

her hourly wage for bagging groceries times her hours spent bagging groceries

her hours spent bagging groceries times her hourly wage for bagging groceries

The phrases are slightly shaded in the answers above to illustrate that they come from the three blocks. As we can see, this set of rules would be suitable for, for example, the first question of the *Explain in English First* strategy, where the student is asked to describe the expression using a set of English phrases. In a nutshell, answer rules specify for the diagnostic system which block attribute slots to look at when diagnosing the student's answer.

Answer block rules are for generating the answer blocks for the student. Their syntax is similar to the rules shown above. Answer block rules are necessary, because sometimes the `Symbol` attributes of blocks and sometimes English phrases need to be provided for the student. For example, if the student is required to assemble  $5 * g$  or  $g * 5$ , then the following answer block rules would need to be used for the question:

|                    |   |        |
|--------------------|---|--------|
| type = operator    | ➡ | symbol |
| type = variable    | ➡ | symbol |
| type = parenthesis | ➡ | symbol |

This would produce the following set of answer blocks for the student: +, -, \*, /, (, ), g. How are these generated? The process is actually quite simple. Mason scans through the entire block library of the current problem. Each time a rule fires for one of the blocks, that block's specified attribute slot value is then added to the student's answer block listing. The rules above specify that: if a block is an operator, a variable, or a parenthesis, then make its `Symbol` value available for the student.

This concludes the section on questions, which play a key role in Mason's dialogue generation. The following section discusses how questions can be grouped in an orderly fashion in order to form strategies.

### 2.2.3 Strategies and Strategy Sets

Most of the time, asking the student a single question about a problematic partition might not prove to be effective tutoring. If a student forgets to include an essential number in the answer, then one question might be sufficient. For example, let's assume that the inputted answer for the "Debbie" problem is missing the number 30. For this error, the following question could be used:

```
You seemed to have missed an important number in your
expression. According to the problem statement, what is her
total number of hours worked a week?
```

All the student has to do is look up the number 30 in the problem statement, and input it as the answer. This student error is a very simple one that doesn't need intensive tutoring. Most errors, however, come from more complex partitions, and a single question might not be enough to tutor the student effectively. For example, let's take a

look at the  $7 * (30 - g)$  partition of the “Debbie” problem; it represents “her weekly salary for delivering newspapers”. If the student is unable to assemble the expression correctly, then a set of questions need to be generated that will eventually lead the student to the correct answer. This is where strategies come in.

Recall from the Ms. Lindquist section of the previous chapter that strategies are nothing more than an ordered set of question templates. They can be constructed and modified under the *Strategies* tab, where a list of all the previously created question templates is available for the author (see Figure 17). Questions can be added to and removed from strategies at any time; their ordering is what matters. For example, let’s take a look at Ms. Lindquist’s *Concrete Articulation Strategy* for the  $30 - g$  problem partition. The following three questions would be asked for tutoring purposes:

- 1.) Could you tell me the value for the number of hours she spends delivering newspapers if  $g$  is 10?  
**Expected answer: 20**, since  $30 - 10 = 20$
- 2.) Now, show how you calculated that the number of hours she spends delivering newspapers is 20 by writing it out as an expression and using 10 for  $g$ .  
**Expected answer:  $30 - 10$**
- 3.) Now, simply replace 10 with  $g$  in your previous answer to get the final expression for the number of hours she spends delivering newspapers.  
**Expected Answer:  $30 - g$**

Thus, *Concrete Articulation* consists of three questions. Of course, strategies can have a variable number of questions for providing additional help to the student. The author, however, needs to balance how long the student should spend on a particular problem partition. For example, it would be illogical to ask ten questions about  $30 - g$ .

Strategy sets are used for the randomization of strategies. If an Intelligent Tutoring System would only use the *Concrete Articulation* strategy for helping the student, the tutoring process would quickly become tedious. A variety of teaching

methods is necessary in order to make the process a more enjoyable and effective experience. Strategy sets can be constructed under the *Strategies* tab (see Figure 17).

Let's take a look at the following:

- 1.) *Concrete Articulation*
- 2.) *Explain in English First*
- 3.) *Convert the Problem into an Example to Explain*
- 4.) *Introduce a New Variable*

This set has four strategies attached to it. Each time the set is called on, it randomly picks one of the strategies for usage. Would this set be adequate for tutoring student on  $30 - g$ ? The *Introduce a New Variable* strategy might not be helpful, because the expression is too simple. Thus, the fourth strategy should be removed from the set.

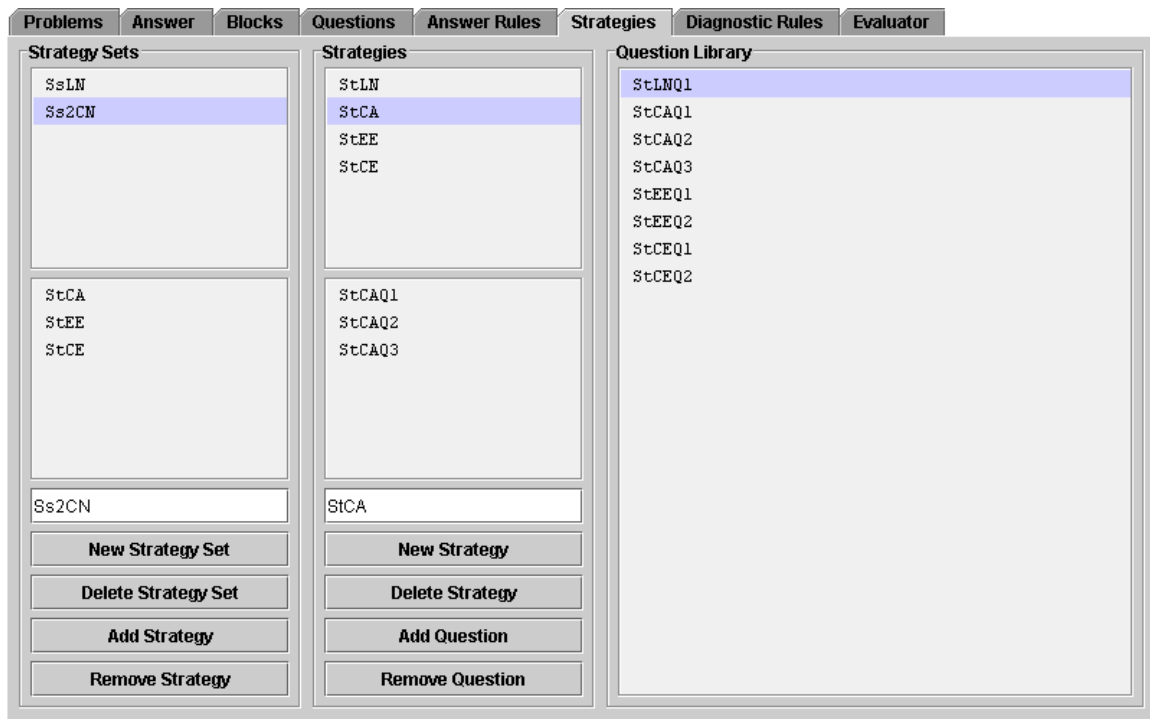


Figure 17 - The graphical user interface for the Strategies tab

It is the author's responsibility to make sure that only applicable strategies are used for tutoring the student. The following section describes diagnostic rules, which determine what strategy sets should be used in certain situations.



## 2.2.4 Diagnostic Rules

Attaching questions to nodes in the answer tree is not an efficient way of creating problems. Complex problems might require hundreds of attached questions, making the ITS construction process tedious. This is why Mason supports diagnostic rules.

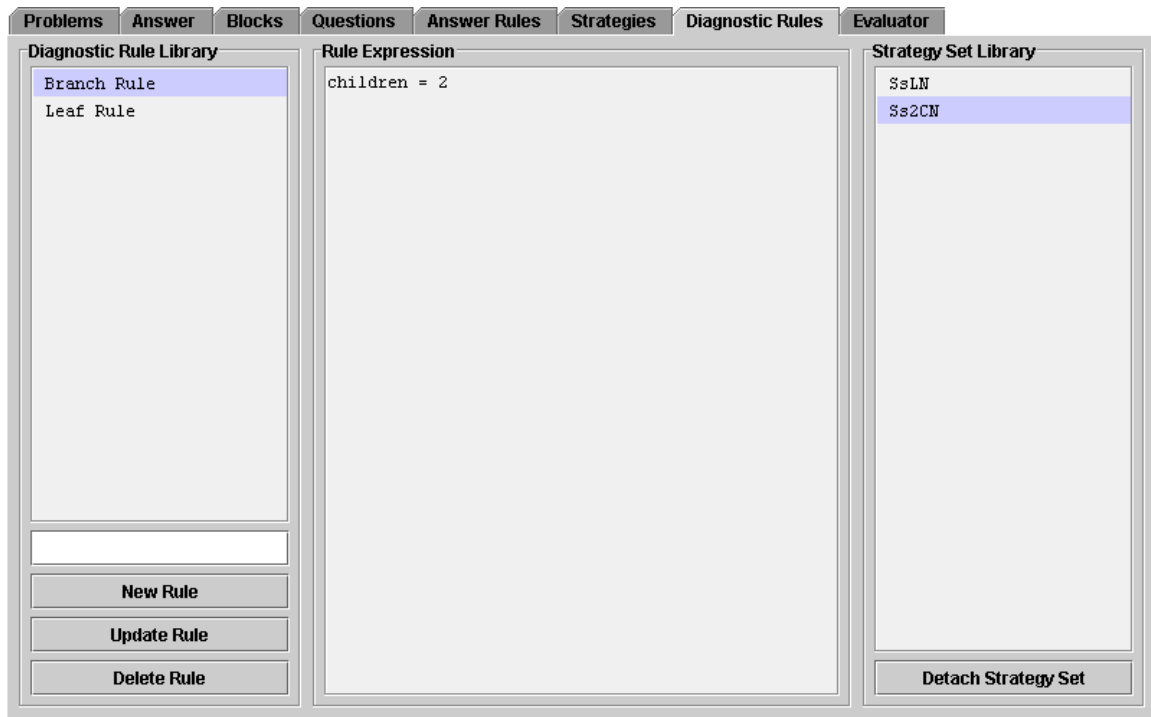


Figure 18 - The graphical user interface for the Diagnostic Rules tab

An evaluator for regular expressions has been built from scratch for Mason to allow for the construction of complex diagnostic rules. These rules are used by the diagnostic system to identify the types of errors in the student answer. As discussed previously, after the diagnostic system finds a faulty node, it checks which diagnostic rule fires for it. Each rule has a strategy set bound to it; therefore, the strategy set of the fired rule is called upon for dialogue generation.

These rules are mainly written for checking `Symbol` attributes of blocks in the answer tree and to check if the student got certain portions of the answer tree correctly.

Yet, some really powerful rules can be written, especially since Mason can evaluate complex regular expressions. Let's take a look at an example for detecting the incorrect usage of the commutative property. Although the student is using the correct variable, number, and operator in the answer, they are assembled incorrectly. For example, instead of  $30 - g$ , the student inputs  $g - 30$ . Let's take a look at the rule itself:

```
children = 2
<and> symbol = -
<and> child 0 -> correct = true
<and> child 1 -> correct = true
```

This rule will only fire if the commutativity is not used properly for the  $-$  operator. It specifies that if the faulty node has two children, and both of them have been answered correctly, then fire the rule. This syntax, however, is not the best solution, since the same commutative law applies to division as well. Although a second rule could be written for division, a single one is sufficient for including both operators. Let's take a look at the following, which is a slightly modified version of the rule above:

```
children = 2
<and> commutative = false
<and> child 0 -> correct = true
<and> child 1 -> correct = true
```

This time, we simply check for the commutativity of the block, making the rule less specific. So far, only the usage of the `<and>` operator has been demonstrated. Mason also supports `<or>` and embedded regular expression. Let's rewrite the commutativity rule to illustrate these elements as well:

```
children = 2
<and> (
    symbol = -
    <or> symbol = /
)
<and> child 0 -> correct = true
<and> child 1 -> correct = true
```

One small technicality to address is the fact that multiple depths of nodes can be reached with a single diagnostic rule. In the example above, the `correct` property of `child 0` was checked. It is actually possible to check a child of that child node as well. In fact, Mason allows the author to write rules that can analyze the answer tree in any depth. For example, “`child 0 -> child 1 -> symbol = 60`” would be a valid regular expression. The problem is that in order to have such a regular expression, the author first needs to verify in the rule that `child 0 -> child 1` actually exists. Otherwise, errors might occur. The complete error-proof rule would look like this:

```
children = 2
<and> child 0 -> children = 2
<and> child 0 -> child 1 -> symbol = 60
```

In sum, diagnostic rules are used by the diagnostic system to determine the type of the error in the student answer for the current faulty block. Once a rule fires, the strategy set bound to it is used for dialogue generation. For example, the diagnostic rule for detecting commutative errors would have a strategy set bound to it, whose strategies would tutor the student on the commutative property. The following chapter illustrates that the replica of Ms. Lindquist actually has some relatively simple diagnostic rules for the four strategies.

## **2.3 Chapter Conclusions**

This chapter described Mason’s architecture in detail. It illustrated how the hierarchical domain model and the diagnostic engine collaborate in order to produce realistic dialogue. The domain model stores all of the ITS data, including question templates, rules, teaching strategies, and pedagogical material. The diagnostic system pinpoints the incorrect portions of the student’s answer, and applies the appropriate

teaching strategies from the domain model to those portions in order to assist the student in finding the correct answer.

This chapter also shows that, due to the nature of its data representation, Mason is optimized for constructing algebraic ITSs. Algebraic expressions are relatively straightforward to represent hierarchically. The ultimate goal of Mason is to become a domain-independent authoring tool. This requires numerous enhancements to not only the authoring tool's architecture, but also its user interface. For example, Mason could be used to write ITSs for programming languages. Currently, the learning environment is so primitive, that the student would not be able to enter chunks of code into the system. The effectiveness of the produced ITSs depends entirely on how creatively they are assembled. Only complex, well-constructed domain models generate realistic dialogue.

By now, the reader should have a clear understanding of how Mason operates. The following chapter describes the evaluation process of the authoring tool, which consists of the reconstruction of an already existing ITS called Ms. Lindquist. The chapter elaborates on how the domain model is constructed in Mason in order to emulate Ms. Lindquist's capabilities as closely as possible.

## Chapter 3: Evaluation

The most illustrative way to evaluate an authoring tool, such as Mason, is by using it to construct a copy of an already existing ITS. The previous chapter pointed out that Mason is best suited for mathematical ITSs, since algebraic expressions can be represented hierarchically. Thus, Ms. Lindquist, the algebra ITS described in Chapter 1, was chosen for this task. Due to its complexity and lengthy development time of roughly two years, it was a logical candidate for the control in Mason's evaluation process. Such an evaluation determines the length of the construction process, and how accurately the clone produced by the authoring tool can emulate the original.

Ms. Lindquist, however, was not reproduced in its entirety for this thesis. The tutoring system contains a myriad of hardcoded problems that are unnecessary for the evaluation process. Additional problems can be added to Ms. Lindquist in form of data files, which take a significant amount of time to create. Therefore, the cloning of the Ms. Lindquist only encompassed the reconstruction of the ITS's technical architecture. Of course, a few sample problems have also been assembled to demonstrate the functionality of the replicated architecture.

The first section of this chapter elaborates on how the Ms. Lindquist strategies and a sample problem were constructed. Strategies, as described in Chapter 1, are the architectural components responsible for dialog generation; therefore, they need to be reconstructed as accurately as possible. After all, the most logical method for determining the accuracy of the reproduction process is by comparing the dialog segments generated by Ms. Lindquist and its replica. The assembly of the sample problem and the strategies

is a partial overview of some of the concepts discussed in Chapter 2; therefore, the actual construction process is not described in detail.

Next, the chapter analyzes the length of the replication process. Mason can in fact be used to reproduce Ms. Lindquist in a significantly shorter period of time. Can the replica diagnose the student's answer as effectively as Ms. Lindquist? Can it produce the same quality of dialogue? The final section provides the answers to these questions and elaborates on the advantages and disadvantages of the Mason architecture.

### ***3.1 The Reconstruction of Ms. Lindquist***

In order to reproduce Ms. Lindquist, it was necessary to identify exactly what components had to be created in order to generate output that was similar in content to that of the ITS. It was observed in Chapter 1 that the dialogue in Ms. Lindquist is generated using a set of templates grouped together in an orderly fashion to form strategies. Before reproducing the strategies, let's take a look at the fully constructed "Debbie" problem. It's much easier to understand what dialogue will be generated by the templates if we know all the attribute slot values of the blocks in the answer tree.

This section will merely display the final version of the Debbie problem. After all, problem construction is explained in Chapter 2. Let's take a look at the initial problem components: the problem statement, and the initial question.

**Problem Statement:** Debbie has two jobs over the summer. At one job, she bags groceries at Giant Eagle and gets paid 5 dollars an hour. At the other job she delivers newspapers and gets paid 7 dollars an hour. She works a total of 30 hours a week, and she works  $g$  hours bagging groceries. Write an expression for the total amount she earns a week.

**Initial Question:** Please write an expression for her total pay per week. (You need to create additional number blocks if necessary.)

The problem has the answer tree structure shown in Figure 13, which is assembled in Mason and has the corresponding blocks bound to its nodes. The blocks play a key role in the dialogue generation process, because it's their slots' values that get inserted into the question templates. Thus, it's necessary to ensure that every block has all the attribute slots required by the strategies in order prevent errors from occurring.






Some errors are handled automatically by the system. For example, if the author tries to access a non-existent attribute value slot of a block, the string `<undefined>` will be returned and the system will remain stable. However, if the author does not provide the correct number of parameters in a template or a diagnostic rule, Java exceptions might occur. Mason attempts to provide feedback regarding most of the errors, but exceptions can cause the system to become unstable.

For the "Debbie" problem, Figure 10 illustrates that the problem requires the creation of eleven answer tree blocks. We need to take into account that two different blocks are necessary for the `*` operator, since they have different values for their `Description` slots. The block for the variable `g` can be used twice. Although it doesn't appear in the answer tree, an additional block should be created for the `/` operator as well. This is used when answer blocks are generated for the student. It's not enough if only `+`, `-`, and `*` are available for the student. The `/` operator should be displayed as well, even if it's not necessary for the problem. Let's take a look at the blocks and their slot values.

|          |                     |                                       |
|----------|---------------------|---------------------------------------|
| <b>5</b> | <b>Symbol:</b>      | 5                                     |
|          | <b>Type:</b>        | number                                |
|          | <b>Description:</b> | her hourly wage for bagging groceries |
|          | <b>Ignore:</b>      | false                                 |
|          | <b>Commutative:</b> | false                                 |

|           |  |
|-----------|--|
| <b>7</b>  | <p><b>Symbol:</b> 7</p> <p><b>Type:</b> number</p> <p><b>Description:</b> her hourly wage for delivering newspapers</p> <p><b>Ignore:</b> false</p> <p><b>Commutative:</b> false</p> <p><b>Variable:</b> 7</p>   |
| <b>30</b> | <p><b>Symbol:</b> 30</p> <p><b>Type:</b> number</p> <p><b>Description:</b> her total number of hours worked a week</p> <p><b>Ignore:</b> false</p> <p><b>Commutative:</b> false</p>  |
| <b>g</b>  | <p><b>Symbol:</b> g</p> <p><b>Type:</b> variable</p> <p><b>Description:</b> her hours spent bagging groceries</p> <p><b>Ignore:</b> false</p> <p><b>Commutative:</b> false</p> <p><b>Numerical:</b> 10</p>   |
| <b>+</b>  | <p><b>Symbol:</b> +</p> <p><b>Type:</b> operator</p> <p><b>Description:</b> Debbie's total weekly salary</p> <p><b>Ignore:</b> false</p> <p><b>Commutative:</b> true</p> <p><b>Value:</b> plus</p> <p><b>Numerical:</b> 190</p>                          |
| <b>-</b>  | <p><b>Symbol:</b> -</p> <p><b>Type:</b> operator</p> <p><b>Description:</b> the number of hours she spends delivering newspapers</p> <p><b>Ignore:</b> false</p> <p><b>Commutative:</b> false</p> <p><b>Value:</b> minus</p> <p><b>Numerical:</b> 20</p> |
| <b>*</b>  | <p><b>Symbol:</b> *</p> <p><b>Type:</b> operator</p> <p><b>Description:</b> her weekly salary for bagging groceries</p> <p><b>Ignore:</b> false</p> <p><b>Commutative:</b> true</p> <p><b>Value:</b> times</p> <p><b>Numerical:</b> 50</p>               |



|   |   |
|---|---|
|    | <b>Symbol:</b> *<br><b>Type:</b> operator<br><b>Description:</b> her weekly salary for delivering newspapers<br><b>Ignore:</b> false<br><b>Commutative:</b> false<br><b>Value:</b> times<br><b>Numerical:</b> 140<br><b>Variable:</b> B |
|    | <b>Symbol:</b> /<br><b>Type:</b> operator<br><b>Description:</b> <ignore><br><b>Ignore:</b> false<br><b>Commutative:</b> false<br><b>Value:</b> divided by  |
|    | <b>Symbol:</b> <ignore><br><b>Type:</b> connector<br><b>Description:</b> the number of hours she spends delivering newspapers<br><b>Ignore:</b> true<br><b>Commutative:</b> false<br><b>Variable:</b> C                                 |
|  | <b>Symbol:</b> (<br><b>Type:</b> parenthesis<br><b>Description:</b> <ignore><br><b>Ignore:</b> true<br><b>Commutative:</b> false  |
|  | <b>Symbol:</b> )<br><b>Type:</b> parenthesis<br><b>Description:</b> <ignore><br><b>Ignore:</b> true<br><b>Commutative:</b> false  |

Most of the attribute slot values, such as `Symbol`, have been discussed in Chapter 2; however, the reader might not understand the necessity of the dynamic attributes, which are displayed in italic. In order to understand why they are present, let's take a look at the replicas of Ms. Lindquist's strategies and see how they use the attribute slots. Examples are given for each of the strategies' templates as well, using the root + node in the "Debbie" problem's answer tree as the current faulty node. The examples include

fully generated questions from each of the question phrases. A possible answer combination to the questions and their dynamic answer block listing are shown as well.

| <b>Strategy 1</b>            |   |
|------------------------------|---|
| <b>Concrete Articulation</b> |   |
| <b>Question 1</b>            |   |
| <b>Phrase 1</b>              | <p>Could you tell me the value of <code>&lt;m&gt; node -&gt; description &lt;/m&gt;</code> if <code>&lt;list&gt; type = variable symbol is numerical and &lt;/list&gt;</code> <code>&lt;ns&gt;</code> ?</p> <p><b>Example:</b> <i>Could you tell me the value of Debbie's total weekly salary if g is 10?</i></p>   |
| <b>Phrase 2</b>              | <p>If <code>&lt;list&gt; type = variable symbol is numerical and &lt;/list&gt;</code> <code>&lt;ns&gt;</code> , then the value of <code>&lt;m&gt; node -&gt; description &lt;/m&gt;</code> is actually <code>&lt;m&gt; node -&gt; numerical &lt;/m&gt;</code> <code>&lt;ns&gt;</code> . So, please input: <code>&lt;m&gt; node -&gt; numerical &lt;/m&gt;</code> <code>&lt;ns&gt;</code> .</p> <p><b>Example:</b> <i>If g is 10, then the value of Debbie's total weekly salary is actually 190. So, please input: 190.</i></p> |
| <b>Phrase 3</b>              | <p>Please input: <code>&lt;m&gt; node -&gt; numerical &lt;/m&gt;</code> <code>&lt;ns&gt;</code> .</p> <p><b>Example:</b> <i>Please input: 190.</i></p>  |
| <b>Depth</b>                 | 0   |
| <b>Rules</b>                 | <p><b>Answer Rules:</b></p> <p>type = operator                      ➡            numerical</p> <p><b>Answer Block Rules:</b></p> <p>type = operator                      ➡            symbol</p> <p>type = variable                      ➡            symbol</p> <p>type = parenthesis                  ➡            symbol</p>   |
| <b>Answer</b>                | 190   |
| <b>Blocks</b>                | <code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>g</code> , <code>(</code> , <code>)</code>  |
| <b>Question 2</b>            |   |
| <b>Phrase 1</b>              | <p>Show how you calculated that <code>&lt;m&gt; node -&gt; description &lt;/m&gt;</code> is <code>&lt;m&gt; node -&gt; numerical &lt;/m&gt;</code> by writing it out as an expression using numbers and <code>&lt;list&gt; type = variable numerical for symbol and &lt;/list&gt;</code> <code>&lt;ns&gt;</code> .</p> <p><b>Example:</b> <i>Show how you calculated that Debbie's total weekly salary is 190 by writing it out as an expression using numbers and 10 for g.</i></p>  |
| <b>Phrase 2</b>              | <p>If <code>&lt;list&gt; type = variable symbol is numerical and &lt;/list&gt;</code> <code>&lt;ns&gt;</code> , then the expression for <code>&lt;m&gt; node -&gt; description &lt;/m&gt;</code> is actually <code>&lt;m&gt; node -&gt; &lt;structure&gt; &lt;/m&gt;</code> <code>&lt;ns&gt;</code> <code>&lt;ns&gt;</code> . So, please input that.</p> <p><b>Example:</b> <i>If g is 10, then the expression for Debbie's</i></p>   |

|                   |   |
|-------------------|---|
|                   | total weekly salary is actually $5 * 10 + 7 * ( 30 - 10 )$ . So, please input that.   |
| <b>Phrase 3</b>   | Please input: <m> node -> <structure> </m> <ns> <ns> .<br><b>Example:</b> Please input: $5 * 10 + 7 * ( 30 - 10 )$ .  |
| <b>Depth</b>      | <max>   |
| <b>Rules</b>      | <b>Answer Rules:</b><br>type = variable                    ➡        numerical<br>symbol = <any>                    ➡        symbol<br><br><b>Answer Block Rules:</b><br>type = operator                    ➡        symbol<br>type = variable                    ➡        symbol<br>type = parenthesis                ➡        symbol |
| <b>Answer</b>     | 5 * 10 + 7 * ( 30 - 10 )  |
| <b>Blocks</b>     | +, -, *, /, g, (, )   |
| <b>Question 3</b> |   |
| <b>Phrase 1</b>   | Now, simply replace <list> type = variable numerical with symbol and </list> in your expression to get the <m> node -> description </m> <ns> .<br><b>Example:</b> Now, simply replace 10 with g in your expression to get Debbie's total weekly.  |
| <b>Phrase 2</b>   | Please input: <m> node -> <structure> </m> <ns> <ns> .<br><b>Example:</b> Please input: $5 * g + 7 * ( 30 - g )$ .  |
| <b>Depth</b>      | <max>   |
| <b>Rules</b>      | <b>Answer Rules:</b><br>symbol = <any>                    ➡        symbol<br><br><b>Answer Block Rules:</b><br>type = operator                    ➡        symbol<br>type = variable                    ➡        symbol<br>type = parenthesis                ➡        symbol  |
| <b>Answer</b>     | 5 * g + 7 * ( 30 - g )  |
| <b>Blocks</b>     | +, -, *, /, g, (, )   |

The *Concrete Articulation* strategy is relatively straight-forward, because it always generates the answer block listing for the student using the `Symbol` attributes of blocks. One item to note is *Depth*. The author can specify for each question how deep Mason should diagnose the answer tree relative to the current faulty node. The value for

*Depth* throughout the replicas of the four strategies is usually 0, 1, or  $\langle \text{max} \rangle$ . Other values are possible but unnecessary for our purposes.

Also, keep in mind that *Answer* lists one possible answer combination only. After all,  $5 * g + 7 * (30 - g)$  has many different combinations due to the commutative property. Finally, *Blocks* displays the answer block listing that is dynamically generated for each question. Note, for example, that number blocks are always missing. As discussed previously, it's best to allow the students to create those blocks themselves. This makes the questions seem less like multiple-choice ones. Below are the three remaining strategies, displayed in the same format as the first:

| <b>Strategy 2</b>               |   |                   |   |                   |               |   |             |                 |   |             |                 |   |                   |
|---------------------------------|---|-------------------|---|-------------------|---------------|---|-------------|-----------------|---|-------------|-----------------|---|-------------------|
| <b>Explain in English First</b> |   |                   |   |                   |               |   |             |                 |   |             |                 |   |                   |
| <b>Question 1</b>               |   |                   |   |                   |               |   |             |                 |   |             |                 |   |                   |
| <b>Phrase 1</b>                 | <p>Can you explain how you would find <math>\langle m \rangle</math> node <math>\rightarrow</math> description <math>\langle /m \rangle</math> <math>\langle ns \rangle</math> ? Please complete the following: <math>\langle m \rangle</math> node <math>\rightarrow</math> description <math>\langle /m \rangle</math> is equal to...</p> <p><i>Example:</i> Could you explain how you would find Debbie's total weekly salary? Please complete the following: Debbie's total weekly salary is equal to...</p>  |                   |   |                   |               |   |             |                 |   |             |                 |   |                   |
| <b>Phrase 2</b>                 | <p>Try to find a relationship between <math>\langle m \rangle</math> child 0 <math>\rightarrow</math> description <math>\langle /m \rangle</math> and <math>\langle m \rangle</math> child 1 <math>\rightarrow</math> description <math>\langle /m \rangle</math> <math>\langle ns \rangle</math> .</p> <p><i>Example:</i> Try to find a relationship between her weekly salary for bagging groceries and her weekly salary for delivering newspapers.</p>  |                   |   |                   |               |   |             |                 |   |             |                 |   |                   |
| <b>Phrase 3</b>                 | <p>Please input: <math>\langle m \rangle</math> node <math>\rightarrow</math> <math>\langle structure \rangle</math> <math>\langle /m \rangle</math> <math>\langle ns \rangle</math> <math>\langle ns \rangle</math> .</p> <p><i>Example:</i> Please input: her weekly salary for bagging groceries plus her weekly salary for delivering newspapers.</p>   |                   |   |                   |               |   |             |                 |   |             |                 |   |                   |
| <b>Depth</b>                    | 1   |                   |   |                   |               |   |             |                 |   |             |                 |   |                   |
| <b>Rules</b>                    | <p><b>Answer Rules:</b></p> <table style="width: 100%; border: none;"> <tr> <td style="width: 40%;">type = operator</td> <td style="width: 10%; text-align: center;">➡</td> <td style="width: 50%;">value description</td> </tr> <tr> <td>type = number</td> <td style="text-align: center;">➡</td> <td>description</td> </tr> <tr> <td>type = variable</td> <td style="text-align: center;">➡</td> <td>description</td> </tr> </table> <p><b>Answer Block Rules:</b></p> <table style="width: 100%; border: none;"> <tr> <td style="width: 40%;">type = operator</td> <td style="width: 10%; text-align: center;">➡</td> <td style="width: 50%;">value description</td> </tr> </table> | type = operator   | ➡ | value description | type = number | ➡ | description | type = variable | ➡ | description | type = operator | ➡ | value description |
| type = operator                 | ➡   | value description |   |                   |               |   |             |                 |   |             |                 |   |                   |
| type = number                   | ➡   | description       |   |                   |               |   |             |                 |   |             |                 |   |                   |
| type = variable                 | ➡   | description       |   |                   |               |   |             |                 |   |             |                 |   |                   |
| type = operator                 | ➡   | value description |   |                   |               |   |             |                 |   |             |                 |   |                   |

|                   |  |
|-------------------|--|
|                   | type = number                    ➡        description<br>type = variable                  ➡        description   |
| <b>Answer</b>     | her weekly salary for bagging groceries plus<br>her weekly salary for delivering newspapers  |
| <b>Blocks</b>     | plus, minus, times, divided by,<br>her hourly wage for bagging groceries,<br>her hourly wage for delivering newspapers,<br>her total number of hours worked a week,<br>her hours spent bagging groceries,<br>Debbie's total weekly salary,<br>the number of hours she spends delivering newspapers,<br>her weekly salary for bagging groceries,<br>her weekly salary for delivering newspapers   |
| <b>Question 2</b> |  |
| <b>Phrase 1</b>   | Now, follow that up by writing an expression for <m><br>node -> description </m> <ns> .<br><br><i><b>Example:</b> Now, follow that up by writing an expression<br/>         for Debbie's total weekly.</i>   |
| <b>Phrase 2</b>   | When inputting your expression, try to use the facts<br>that <m> child 0 -> description </m> is <m> child 0 -><br><structure> </m> <ns> and <m> child 1 -> description<br></m> is <m> child 1 -> <structure> </m> <ns> <ns> .<br><br><i><b>Example:</b> When inputting your expression, try to use<br/>         the facts that her weekly salary for bagging groceries<br/>         is <math>5 * g</math> and her weekly salary for delivering<br/>         newspapers is <math>7 * ( 30 - g )</math>.</i> |
| <b>Phrase 3</b>   | Please input: <m> node -> <structure> </m> <ns> <ns> .<br><br><i><b>Example:</b> Please input: <math>5 * g + 7 * ( 30 - g )</math>.</i>  |
| <b>Depth</b>      | <max>  |
| <b>Rules</b>      | <b>Answer Rules:</b><br>symbol = <any>                    ➡        symbol<br><br><b>Answer Block Rules:</b><br>type = operator                    ➡        symbol<br>type = variable                    ➡        symbol<br>type = parenthesis                ➡        symbol   |
| <b>Answer</b>     | $5 * g + 7 * ( 30 - g )$   |
| <b>Blocks</b>     | + , - , * , / , g , ( , )  |

| <b>Strategy 3</b>                                     |  |
|---|--|
| <b>Convert the Problem into an Example to Explain</b> |  |
| <b>Question 1</b>                                     |  |
| <b>Phrase 1</b>                                       | The answer is actually $\langle m \rangle$ node $\rightarrow$ $\langle structure \rangle$ $\langle /m \rangle$ $\langle ns \rangle$ $\langle ns \rangle$ , so please input that.<br><br><i>Example:</i> The answer is actually $5 * g + 7 * ( 30 - g )$ , so please input that.  |
| <b>Phrase 2</b>                                       | Please input: $\langle m \rangle$ node $\rightarrow$ $\langle structure \rangle$ $\langle /m \rangle$ $\langle ns \rangle$ $\langle ns \rangle$ .<br><br><i>Example:</i> Please input: $5 * g + 7 * ( 30 - g )$ .  |
| <b>Depth</b>  | $\langle max \rangle$  |
| <b>Rules</b>  | <b>Answer Rules:</b><br>symbol = $\langle any \rangle$ $\Rightarrow$ symbol<br><br><b>Answer Block Rules:</b><br>type = operator $\Rightarrow$ symbol<br>type = variable $\Rightarrow$ symbol<br>type = parenthesis $\Rightarrow$ symbol   |
| <b>Answer</b>   | $5 * g + 7 * ( 30 - g )$   |
| <b>Blocks</b>   | $+$ , $-$ , $*$ , $/$ , $g$ , $($ , $)$  |
| <b>Question 2</b>                                     |  |
| <b>Phrase 1</b>                                       | Let me ask you a few questions to help you understand the answer. What does $\langle m \rangle$ child 0 $\rightarrow$ $\langle structure \rangle$ symbol $\langle /m \rangle$ $\langle ns \rangle$ represent in the equation?<br><br><i>Example:</i> Let me ask you a few questions to help you understand the answer. What does $5 * g$ represent in the equation?                      |
| <b>Phrase 2</b>                                       | Actually, $\langle m \rangle$ child 0 $\rightarrow$ $\langle structure \rangle$ symbol $\langle /m \rangle$ $\langle ns \rangle$ represents $\langle m \rangle$ child 0 $\rightarrow$ description $\langle /m \rangle$ $\langle ns \rangle$ . So, please input that.<br><br><i>Example:</i> Actually, $5 * g$ represents her weekly salary for bagging groceries. So, please input that. |
| <b>Phrase 3</b>                                       | Please input: $\langle m \rangle$ child 0 $\rightarrow$ description $\langle /m \rangle$ $\langle ns \rangle$ .<br><br><i>Example:</i> Please input: her weekly salary for bagging groceries.  |
| <b>Depth</b>  | 0  |
| <b>Rules</b>  | <b>Answer Rules:</b><br>child 0 $\rightarrow$ symbol = $\langle any \rangle$ $\Rightarrow$ description<br><br><b>Answer Block Rules:</b><br>type = operator $\Rightarrow$ description<br>type = number $\Rightarrow$ description<br>type = variable $\Rightarrow$ description  |
| <b>Answer</b>   | her weekly salary for bagging groceries  |

|                   |  |
|-------------------|--|
| <b>Blocks</b>     | her hourly wage for bagging groceries,<br>her hourly wage for delivering newspapers,<br>her total number of hours worked a week,<br>her hours spent bagging groceries,<br>Debbie's total weekly salary,<br>the number of hours she spends delivering newspapers,<br>her weekly salary for bagging groceries,<br>her weekly salary for delivering newspapers  |
| <b>Question 3</b> |  |
| <b>Phrase 1</b>   | Now, what does <code>&lt;m&gt; child 1 -&gt; &lt;structure&gt; symbol &lt;/m&gt;</code><br><code>&lt;ns&gt;</code> represent in the answer <code>&lt;m&gt; node -&gt; &lt;structure&gt;</code><br><code>symbol &lt;/m&gt; &lt;ns&gt; &lt;ns&gt; ?</code><br><br><i><b>Example:</b> Now, what does <math>7 * ( 30 - g )</math> represent in<br/>the answer <math>5 * g + 7 * ( 30 - g )</math>?</i> |
| <b>Phrase 2</b>   | Actually, <code>&lt;m&gt; child 1 -&gt; &lt;structure&gt; symbol &lt;/m&gt; &lt;ns&gt;</code><br>represents <code>&lt;m&gt; child 1 -&gt; description &lt;/m&gt; &lt;ns&gt; .</code> So,<br>please input that.<br><br><i><b>Example:</b> Actually, <math>7 * ( 30 - g )</math> represents her<br/>weekly salary for delivering newspapers. So, please<br/>input that.</i>                          |
| <b>Phrase 3</b>   | Please input: <code>&lt;m&gt; child 1 -&gt; description &lt;/m&gt; &lt;ns&gt; .</code><br><br><i><b>Example:</b> Please input: her weekly salary for<br/>delivering newspapers.</i>  |
| <b>Depth</b>      | 0  |
| <b>Rules</b>      | <b>Answer Rules:</b><br><code>child 1 -&gt; symbol = &lt;any&gt;      ➡      description</code><br><br><b>Answer Block Rules:</b><br><code>type = operator                      ➡      description</code><br><code>type = number                         ➡      description</code><br><code>type = variable                        ➡      description</code>                                       |
| <b>Answer</b>     | her weekly salary for delivering newspapers  |
| <b>Blocks</b>     | her hourly wage for bagging groceries,<br>her hourly wage for delivering newspapers,<br>her total number of hours worked a week,<br>her hours spent bagging groceries,<br>Debbie's total weekly salary,<br>the number of hours she spends delivering newspapers,<br>her weekly salary for bagging groceries,<br>her weekly salary for delivering newspapers  |
| <b>Question 4</b> |  |
| <b>Phrase 1</b>   | Finally, what does <code>&lt;m&gt; node -&gt; &lt;structure&gt; symbol &lt;/m&gt;</code>   |

|                 |   |                   |   |                   |               |   |             |                 |   |             |                 |   |                   |               |   |             |                 |   |             |
|-----------------|---|-------------------|---|-------------------|---------------|---|-------------|-----------------|---|-------------|-----------------|---|-------------------|---------------|---|-------------|-----------------|---|-------------|
|                 | <p>represent? (I am not looking for the fact that it is &lt;m&gt; node -&gt; description &lt;/m&gt; &lt;ns&gt;, but rather the relationship between the two entities that we just discussed.)</p> <p><b>Example:</b> Finally, what does <math>5 * g + 7 * ( 30 - g )</math> represent? (I am not looking for the fact that it is Debbie's total weekly salary, but rather the relationship between the two entities that we just discussed.)</p>  |                   |   |                   |               |   |             |                 |   |             |                 |   |                   |               |   |             |                 |   |             |
| <b>Phrase 2</b> | <p>Actually, the answer I was looking for is that &lt;m&gt; node -&gt; &lt;structure&gt; symbol &lt;/m&gt; &lt;ns&gt; represents &lt;m&gt; node -&gt; structure &lt;/m&gt; &lt;ns&gt; &lt;ns&gt; . So, please input that.</p> <p><b>Example:</b> Actually, the answer I was looking for is that <math>5 * g + 7 * ( 30 - g )</math> represents her weekly salary for bagging groceries plus her weekly salary for delivering newspapers. So, please input that.</p>   |                   |   |                   |               |   |             |                 |   |             |                 |   |                   |               |   |             |                 |   |             |
| <b>Phrase 3</b> | <p>Please input: &lt;m&gt; node -&gt; &lt;structure&gt; &lt;/m&gt; &lt;ns&gt; &lt;ns&gt; .</p> <p><b>Example:</b> Please input: her weekly salary for bagging groceries plus her weekly salary for delivering newspapers.</p>   |                   |   |                   |               |   |             |                 |   |             |                 |   |                   |               |   |             |                 |   |             |
| <b>Depth</b>    | 1   |                   |   |                   |               |   |             |                 |   |             |                 |   |                   |               |   |             |                 |   |             |
| <b>Rules</b>    | <p><b>Answer Rules:</b></p> <table> <tr> <td>type = operator</td> <td>➡</td> <td>value description</td> </tr> <tr> <td>type = number</td> <td>➡</td> <td>description</td> </tr> <tr> <td>type = variable</td> <td>➡</td> <td>description</td> </tr> </table> <p><b>Answer Block Rules:</b></p> <table> <tr> <td>type = operator</td> <td>➡</td> <td>value description</td> </tr> <tr> <td>type = number</td> <td>➡</td> <td>description</td> </tr> <tr> <td>type = variable</td> <td>➡</td> <td>description</td> </tr> </table> | type = operator   | ➡ | value description | type = number | ➡ | description | type = variable | ➡ | description | type = operator | ➡ | value description | type = number | ➡ | description | type = variable | ➡ | description |
| type = operator | ➡   | value description |   |                   |               |   |             |                 |   |             |                 |   |                   |               |   |             |                 |   |             |
| type = number   | ➡   | description       |   |                   |               |   |             |                 |   |             |                 |   |                   |               |   |             |                 |   |             |
| type = variable | ➡   | description       |   |                   |               |   |             |                 |   |             |                 |   |                   |               |   |             |                 |   |             |
| type = operator | ➡   | value description |   |                   |               |   |             |                 |   |             |                 |   |                   |               |   |             |                 |   |             |
| type = number   | ➡   | description       |   |                   |               |   |             |                 |   |             |                 |   |                   |               |   |             |                 |   |             |
| type = variable | ➡   | description       |   |                   |               |   |             |                 |   |             |                 |   |                   |               |   |             |                 |   |             |
| <b>Answer</b>   | <p>her weekly salary for bagging groceries plus<br/>her weekly salary for delivering newspapers</p>   |                   |   |                   |               |   |             |                 |   |             |                 |   |                   |               |   |             |                 |   |             |
| <b>Blocks</b>   | <p>plus, minus, times, divided by,<br/>her hourly wage for bagging groceries,<br/>her hourly wage for delivering newspapers,<br/>her total number of hours worked a week,<br/>her hours spent bagging groceries,<br/>Debbie's total weekly salary,<br/>the number of hours she spends delivering newspapers,<br/>her weekly salary for bagging groceries,<br/>her weekly salary for delivering newspapers</p>   |                   |   |                   |               |   |             |                 |   |             |                 |   |                   |               |   |             |                 |   |             |



| <b>Strategy 4</b>               |   |
|---------------------------------|---|
| <b>Introduce a New Variable</b> |   |
| <b>Question 1</b>               |   |
| <b>Phrase 1</b>                 | <p>If <math>\langle m \rangle</math> child 0 <math>\rightarrow</math> variable <math>\langle /m \rangle</math> was <math>\langle m \rangle</math> child 0 <math>\rightarrow</math> description <math>\langle /m \rangle</math> and <math>\langle m \rangle</math> child 1 <math>\rightarrow</math> variable <math>\langle /m \rangle</math> was <math>\langle m \rangle</math> child 1 <math>\rightarrow</math> description <math>\langle /m \rangle</math> <math>\langle ns \rangle</math> , then what would be <math>\langle m \rangle</math> node <math>\rightarrow</math> description <math>\langle /m \rangle</math> <math>\langle ns \rangle</math> ?</p> <p><b>Example:</b> If A was her weekly salary for bagging groceries and B was her weekly salary for delivering newspapers, then what would be Debbie's total weekly salary?</p> |
| <b>Phrase 2</b>                 | <p>Remember that <math>\langle m \rangle</math> node <math>\rightarrow</math> description <math>\langle /m \rangle</math> equals to <math>\langle m \rangle</math> child 0 <math>\rightarrow</math> description <math>\langle /m \rangle</math> <math>\langle m \rangle</math> node <math>\rightarrow</math> value <math>\langle /m \rangle</math> <math>\langle m \rangle</math> child 1 <math>\rightarrow</math> description <math>\langle /m \rangle</math> <math>\langle ns \rangle</math> . Please try again.</p> <p><b>Example:</b> Remember that Debbie's total weekly salary equals to her weekly salary for bagging groceries plus her weekly salary for delivering newspapers. Please try again.</p>  |
| <b>Phrase 3</b>                 | <p>Please input: <math>\langle m \rangle</math> node <math>\rightarrow</math> <math>\langle structure \rangle</math> <math>\langle /m \rangle</math> <math>\langle ns \rangle</math> <math>\langle ns \rangle</math> .</p> <p><b>Example:</b> Please input: A + B.</p>  |
| <b>Depth</b>                    | 1   |
| <b>Rules</b>                    | <p><b>Answer Rules:</b></p> <p>depth = 0 <math>\Rightarrow</math> value</p> <p>symbol = <math>\langle any \rangle</math> <math>\Rightarrow</math> variable</p> <p><b>Answer Block Rules:</b></p> <p>type = operator <math>\Rightarrow</math> value variable</p> <p>symbol = <math>\langle any \rangle</math> <math>\Rightarrow</math> variable</p>  |
| <b>Answer</b>                   | $A + B$   |
| <b>Blocks</b>                   | $+$ , $-$ , $*$ , $/$ , A, B, C   |
| <b>Question 2</b>               |   |
| <b>Phrase 1</b>                 | <p>Now, can you state this in terms of the numbers given in the problem?</p> <p><b>Example:</b> Now, can you state this in terms of the numbers given in the problem?</p>   |
| <b>Phrase 2</b>                 | <p>Please input: <math>\langle m \rangle</math> node <math>\rightarrow</math> <math>\langle structure \rangle</math> <math>\langle /m \rangle</math> <math>\langle ns \rangle</math> <math>\langle ns \rangle</math> .</p> <p><b>Example:</b> Please input: <math>5 * g + 7 * ( 30 - g )</math>.</p>  |
| <b>Depth</b>                    | $\langle max \rangle$   |
| <b>Rules</b>                    | <p><b>Answer Rules:</b></p> <p>symbol = <math>\langle any \rangle</math> <math>\Rightarrow</math> symbol</p> <p><b>Answer Block Rules:</b></p> <p>type = operator <math>\Rightarrow</math> symbol</p>   |

|               |                        |   |        |
|---------------|------------------------|---|--------|
|               | type = variable        | ➡ | symbol |
|               | type = parenthesis     | ➡ | symbol |
| <b>Answer</b> | 5 * g + 7 * ( 30 - g ) |   |        |
| <b>Blocks</b> | +, -, *, /, g, (, )    |   |        |

As we can observe from the strategies, the question templates do in fact use all of the attribute slots of the defined blocks. For example, while the *Concrete Articulation* strategy never requires the *Value* slot, the *Introduce a New Variable* strategy does. Thus, the additional dynamic slots are essential for the emulation of Ms. Lindquist's strategies. A higher number of attribute slots allows for more flexible dialogue generation, but also adds to the overall ITS development time. Also, each question currently contains at most three question phrases. This is sufficient for the emulation of Ms. Lindquist. The author, however, can add additional phrases for more in-depth hints or suggestions for students.

Finally, please note how the answer blocks provided for the student change from question to question. Sometimes operators and variables are provided for the student; at other times, English phrases are given. Recall that numbers are never provided for the student as answer blocks, unless they are specifically input by the student. For example, for the first question of the *Concrete Articulation* strategy, the student needs to construct a block for *190*. This makes the question seem less like a multiple-choice one.

Are these strategies sufficient for the emulation of Ms. Lindquist? Actually, one more simple strategy is required. What happens when a specific number or variable is missing from the student answer? It would be illogical if a complex strategy from Ms. Lindquist would get executed for a missing number. Recall that numbers and variables are always leaf nodes in the answer tree; therefore, we can construct a simple strategy that can inquire the student about them. The following set of templates does the job:

| <b>Strategy 5</b>                      |   |
|--|---|
| <b>Inquire about Missing Leaf Node</b> |   |
| <b>Question 1</b>                      |   |
| <b>Phrase 1</b>                        | <p>You seem to be missing <code>&lt;m&gt; node -&gt; description &lt;/m&gt;</code> in your answer. According to the problem statement, that is <code>&lt;m&gt; node -&gt; description &lt;/m&gt;</code> <code>&lt;ns&gt;</code> ?</p> <p><i><b>Example:</b> You seem to be missing her hourly wage for delivering newspapers in your answer. According to the problem statement, what is her hourly wage for delivering newspapers?</i></p> |
| <b>Phrase 2</b>                        | <p>Actually, <code>&lt;m&gt; node -&gt; description &lt;/m&gt;</code> is <code>&lt;m&gt; node -&gt; symbol &lt;/m&gt;</code> <code>&lt;ns&gt;</code> . So, please input that.</p> <p><i><b>Example:</b> Actually, her hourly wage for delivering newspapers is 5. So, please input that.</i></p>  |
| <b>Phrase 3</b>                        | <p>Please input: <code>&lt;m&gt; node -&gt; symbol &lt;/m&gt;</code> <code>&lt;ns&gt;</code> .</p> <p><i><b>Example:</b> Please input: 5.</i></p>   |
| <b>Depth</b>                           | 0   |
| <b>Rules</b>                           | <p><b>Answer Rules:</b><br/> <code>symbol = &lt;any&gt;</code>                      ➡      <code>symbol</code></p> <p><b>Answer Block Rules:</b><br/> <code>type = variable</code>                      ➡      <code>symbol</code></p>  |
| <b>Answer</b>                          | 5   |
| <b>Blocks</b>                          | g   |

Now, let's take a look at the constructed strategy sets and diagnostic rules. For Ms. Lindquist, this rule base is surprisingly simple! The ITS has over seventy production rules for understanding algebraic expressions. Mason, on the hand, can emulate Ms. Lindquist's dialogue generation features accurately without such production rules. This has its advantages and disadvantages.

The advantage is that only a few rules are required for selecting eligible strategies. The disadvantage is that Mason doesn't understand the material on which it tutors the student. Understanding the student's answer plays a key role in Ms. Lindquist's dialogue generation, since the tutoring system can adept its dialogue to the student's expression. Mason, on the other hand, can only scan the student's answer for possible good answers.

Despite this drawback, Mason is still able to generate dialogue that is reminiscent of Ms. Lindquist's. Now, let's take a look at the diagnostic rules as well as the strategy sets bound to them:

| <b>Rule</b>   | <b>Strategy Set</b>   |
|---|-----------------------|
| children = 0  | Strategy 5            |
| children = 2  | Strategies 1, 2       |
| children = 2<br><and> ( child 0 -> children > 0<br><or> child 1 -> children > 0 ) | Strategies 1, 2, 4    |
| children = 2<br><and> depth = 0   | Strategies 1, 2, 3, 4 |

As we can see, the diagnostic rules don't have much to do with actual mathematics. They simply specify how many children the current faulty node is required to have in order for certain strategies to be available for dialogue generation. More complex rules can be written if the author wishes to define strategies for very specific errors. For example, students often forget to use parentheses in their answers. An entire strategy can be created just for dealing with missing parentheses.

### **3.2 Evaluation of Reconstruction Time**

The purpose of authoring tools is to speed up ITS construction time; thus, it's essential to analyze whether or not this is true for Mason. Ms. Lindquist took roughly a year to create. How much faster can Mason reproduce the ITS? It's best to analyze this by evaluating the construction times of each component separately. The first component to look at is library of strategies, since strategies are the most time-consuming to build. Not only are they responsible for dialogue generation, but also contain answer rules used by the diagnostic system. The table below displays the measures assembly times of the strategies and their corresponding question templates.

| <b>Strategy 1: Concrete Articulation</b>                          |                   |                   |
|---|-------------------|-------------------|
| <b>Question 1</b>   | <b>Question 2</b> | <b>Question 3</b> |
| ~15 minutes   | ~20 minutes       | ~5 minutes        |
| <b>Strategy 2: Explain in English First</b>                       |                   |                   |
| <b>Question 1</b>   | <b>Question 2</b> |                   |
| ~25 minutes   | ~10 minutes       |                   |
| <b>Strategy 3: Convert the Problem into an Example to Explain</b> |                   |                   |
| <b>Question 1</b>   | <b>Question 2</b> | <b>Question 3</b> |
| ~25 minutes   | ~15 minutes       | ~5 minutes        |
| <b>Strategy 4: Introduce a New Variable</b>                       |                   |                   |
| <b>Question 1</b>   | <b>Question 2</b> |                   |
| ~25 minutes   | ~5 minutes        |                   |

We can see that the strategies can in fact be constructed in a relatively short period of time. Strategies and question templates are meant to be universal; thus, it's important to assemble problems in a way so that no problem-specific strategies and additional question templates need to be created.

Now that we observed the data collected on the strategies, let's take a look at how long the construction of the "Debbie" problem took. The assembly of the answer tree and the creation of the blocks took roughly an hour altogether. Keep in mind that before the actual construction process, the template for the Ms. Lindquist version of the problem was already available. Yet, a significant amount of conceptualization was still required, since Ms. Lindquist stores problems differently than Mason. Thus, it's safe to say that the addition of brand new problems takes roughly the same amount of time as the reconstruction of Ms. Lindquist's problems. Problem construction time is relative to problem complexity.

To see if the assembly of additional problems takes shorter, the "Anne" exercise was reconstructed as well. It has a simpler answer tree and fewer blocks, thus the

assembly process took roughly thirty minutes. The reconstruction of the strategies is unnecessary, because the ones for the “Debbie” problem can immediately be used for the “Anne” problem as well. This reusability saves significantly speeds up the construction process.

We need to take into consideration that the measurements only take the component construction time into consideration. It actually took a considerable amount of time to analyze Ms. Lindquist’s architecture and design the question templates for the replica. All in all, it’s safe to say that Ms. Lindquist can be reproduced with Mason in roughly a week. That’s still considerably shorter than a year, which is how long it took build Ms. Lindquist from scratch!

### ***3.3 Evaluation of Reconstruction Accuracy***

Finally, how well does Mason’s replica of Ms. Lindquist fare compared to the original? Are their generated dialogues similar? Do their diagnostic routines lead the student to the right answer in a similar fashion? The most logical way to answer these questions is to input two identical sets of student answers into each of the systems and compare their generated dialogue. Let’s take a look at the dialogue for the Concrete Articulation strategy in Chapter 1, and compare it to what Mason generates for the same exact input values:

**Problem:** Anne is rowing a boat in a lake and is 800 yards from the dock from which she started. She rows back towards the dock at 40 yards per minute for  $m$  minutes and stops to rest. How far is she from the dock now?

**Tutor:** Please write an expression for the distance Anne has left to row.

**Student:**  $800 + 40 * m$

**Tutor:** No. But,  $40 * m$  is correct for the distance rowed so far. Could you tell me the value of the distance Anne has left to row if  $m$  is 3?

**Student:** 680

**Tutor:** Correct. Show how you calculated that the distance rowed so far is 680 by writing it out as an expression using numbers and 3 for  $m$ .

**Student:**  $800 - 40 * 3$

**Tutor:** Correct. Now, simply replace 3 with  $m$  in your expression to get the final answer.

**Student:**  $800 - 40 * m$

**Tutor:** Correct. You solved the entire problem!

Aesthetics aside, the content of the dialogue above is almost identical for Ms. Lindquist's. We can note one major difference, which appears in "Show how you calculated that the distance rowed so far is 680 by writing it out as an expression using numbers and 3 for  $m$ ." Ms. Lindquist actually lists the numbers for the student, whereas Mason only mentions that numbers should be used. The current version of the authoring tool doesn't support the generation of such lists, since similar dialogue can be generated using the system's `<list>` function. Other than this discrepancy, the dialogue is almost identical in content. This is not only true for the *Concrete Articulation* strategy, but for the *Explain in English First* and *Introduce a New Variable* strategies as well.

The only strategy that truly illustrates Mason's limitation is the *Convert the Problem into an Example to Explain* strategy. The problem is that this strategy uses a top-down methodology for tutoring the student, and Mason only supports bottom-up diagnosis. Thus, I was forced to create question templates that would emulate Ms. Lindquist's dialogue as closely as possible. Let's take a look at what Mason generates:

**Problem:** In math class there are 12 more boys than girls. If the number of girls is  $g$ , write an expression for the total number of students in the class.

**Tutor:** Please write an expression for the total number of students in a class.

**Student:**  $2 * g - 12$

**Tutor:** Hmm. No. The answer is actually  $g + (g + 12)$ , so please input that.

**Student:**  $g + (g + 12)$

**Tutor:** Good. Let me ask you a few questions to help you understand the answer. What does  $g$  represent in the equation?

**Student:** the total number of students minus the number of boys

**Tutor:** Hmm. No. Actually,  $g$  represents the number of girls. So, please input that.

**Student:** the number of girls

**Tutor:** Good. Now, what does  $(g + 12)$  represent in the answer  $g + (g + 12)$ ?

**Student:** the number of boys

**Tutor:** Correct. Finally, where does  $g + (g + 12)$  come from? (I am not looking for the fact that it is the total number of students in the class, but rather the relationship between the two entities that we just discussed.)

**Student:** the number of boys plus the number of girls

**Tutor:** Correct. You solved the entire problem!

From the dialogue above, we can observe that the ITS produced by Mason has its limitations. Ms. Lindquist has a major advantage over Mason: it actually understands the algebraic expressions on which it tutors the student. This feature becomes essential when the student inputs a simplified or expanded version of the expression. Mason would simply mark the answer as wrong, since it cannot generate variations of a single mathematical expression or a set of English phrases. Ms. Lindquist, however, can verify that the student's answer is correct either by coming up with the same expression using a set of mathematical rules, or by evaluating the student's answer using a bunch of different numerical values and checking if the results match those of the desired expression for the same set of numerical values. This feature provides additional flexibility for Ms. Lindquist's diagnostic system.



### **3.4 Chapter Conclusion**

This chapter has demonstrated that the replica of Ms. Lindquist accurately emulates the ITS with numerous limitations. The fact that Mason does not understand the domain in which it tutors the student is a huge drawback. Yet, its dialogue generation features match those of the tutoring system. The chapter has also demonstrated that Ms. Lindquist can be reproduced in a significantly shorter period of time using Mason. The ITS's architecture and a couple of sample problems were replicated in roughly a week, whereas Ms. Lindquist's construction, which included both design and implementation, took over a year.

## Chapter 4: Discussion

The goal of this thesis is to create an authoring tool that can significantly reduce the time needed to construct text-based ITSs, which diagnose student answers and tutor students by asking them a series of dynamically generated pedagogical questions. The previous chapter has demonstrated that this goal has been met. Using Mason, it is possible to reproduce a close replica of Ms. Lindquist, the complex ITS chosen for evaluation, in a relatively much short period of time. The previous chapter also pointed out Mason's strengths and shortcomings, which are visible when Mason's dialogue-generation is compared to Ms. Lindquist's. This chapter discusses the authoring tool's limitations, and how it can be improved upon in both the graphical user interface and system architecture departments.

### **4.1 Limitations**

Currently, Mason's hierarchical domain model is relatively rigid, and this results in numerous limitations. Representing the problem answer as a tree has great potential, mainly due to the fact that decision trees can be built! However, due to Mason's rigid tree structure, this is not currently possible. Every single node in the answer tree has to show up in the student's answer in order for the system to mark the answer correct.

In order to implement structures like decisions trees, Mason needs to support <or> nodes. An <or> node gives the diagnostic system a choice of choosing either one of the node's children as a path when comparing the student answer to the answer tree. To illustrate this, let's look at a simple problem. Let's create the answer tree for putting on

socks and shoes. In the tree below, “LS” stands for “Left Sock”, “RS” for “Right Sock”, “LSh” for “Left Shoe”, and “RSh” for “Right Shoe”.

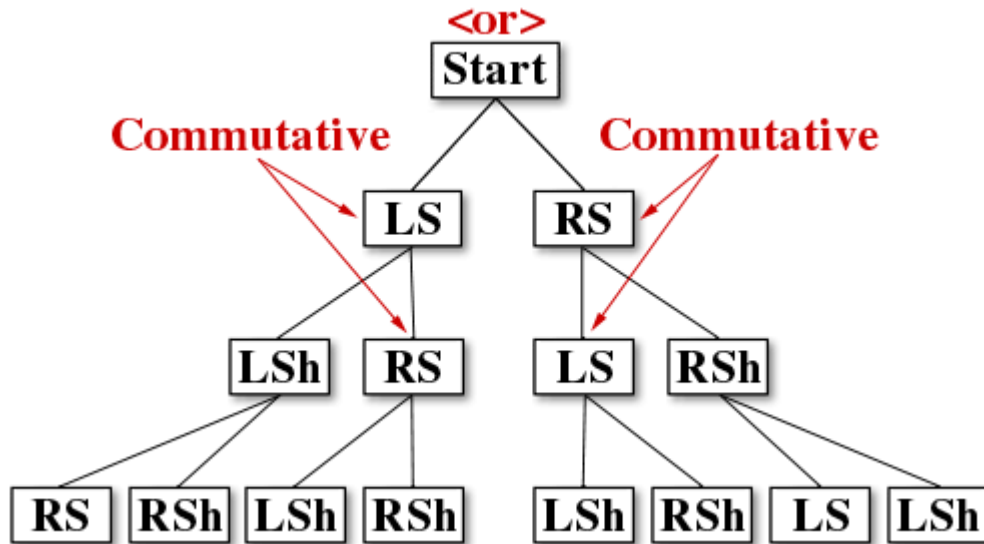


Figure 19 - Answer tree for the "Shoe" problem

As we can see, simply by adding an `<or>` node, which allows the diagnostic system to choose a path, we can create simple decision trees. This not only makes the system more flexible, but also domain-independent! Mason would not be limited to algebra anymore. The problem with Mason’s current implementation is that diagnostic system has to make sure that every node in the answer tree exists in the student’s answer. The `<or>` node would eliminate this rigid limitation.

Also, it has been mentioned that Mason knows nothing about the material on which is tutors the student. This is a major drawback, especially during the diagnostic phase. If the student inputs a simplified version of the desired answer, his or her answer will be marked as incorrect. Mason needs support for **answer tree rules** that can reorder blocks (or combine their attributes) in ways so that more student answers are accepted. This is no small feat, especially if the system doesn’t understand the meaning of the

blocks. Thus, an elementary understanding of numbers is necessary, to which the author can then apply the answer tree rules.

## **4.2 Future Work and Conclusions**

First, let's take a look at Mason's graphical user interface. Initially, a drag-and-drop interface was planned, in which the user would only have a single workspace to work in instead of several areas accessible only by tabs. This interface would have made the construction of blocks and answer trees, both of which would have colorful graphical representations for clarity, much faster and less cumbersome than what the current version of Mason offers. The current interface, however, is quite easy to use once the author gets past the steep learning curve. As illustrated in Chapter 3, it is possible for an experienced Mason user to replicate a complex ITS in a relatively short period of time. Yet, a more user-friendly interface would greatly improve upon the speed and quality of the ITS construction process.

One of the major problems of the user interface is the rule syntax. For example, the question template code `<list> type = variable symbol is numerical and </list>` makes little logical sense even for an expert computer user. Awkward syntax like previous code snippet is partially responsible for Mason's steep learning curve. The following two slight modifications would make the code more readable:

```
<list> if type = variable then print symbol is numerical
using and </list>
<list> if ( type = variable ) then print symbol "is"
numerical using "and" </list>
```

Some of today's sophisticated development tools provide pull-down menus for assembling code fragments. Implementing such a feature would reduce the amount of typing required to construct question templates and rules as well as lessen the

occurrences of typos. Currently, Mason lacks a good error-checking mechanism. Thus, if the user misspells a keyword or forgets to provide the correct number of parameters for a feature, serious errors can occur. Such errors include Java exceptions, which may require the re-launch of Mason. This prevents the tool from being commercially available. A good user interface would support an effective error-detection mechanism.

Now, let's move onto the dialogue generation system of Mason. Aesthetically, one of the authoring tool's flaws is that it lacks customization features for positive and negative feedback. Aesthetics are important; a good author can get really grab the attention of the student with catchy feedback that appears before template-generated dialogue. Right now, Mason has a library of short phrases to randomly pick from, but this might get repetitive for students who use the system for lengthy periods of time. Finally, the authoring tool should be accompanied by tutoring software with a sophisticated student model. Right now, the only way to execute problems in Mason is to do it under the *Evaluator* tab.

Mason has satisfied the goal of this thesis. It's a solid authoring tool for constructing text-based ITSs that are able to diagnose student answers and generate pedagogical dialogue accordingly. With numerous major upgrades to its graphical user interface and internal architecture, it can truly become a flexible, domain-independent authoring tool for making the complex process of ITS construction easier.

## References

- Bloom, B. S. (1984). The 2-Stigma problem: The search for methods of group instruction as effective as one-on-one tutoring. *Educational Researcher*, 13, 4-16.
- Bruner, J. (1966). *Toward a theory of instruction*. Cambridge, MA: Harvard University Press.
- Ginsburg, H., & Opper, S. (1979). *Piaget's theory of intellectual development*. Englewood Cliffs, NJ: Prentice Hall.
- Heffernan, N. T. (2001). *Intelligent tutoring systems are forgetting the tutor: adding a cognitive model of human tutors*. Dissertation. Computer Science Department, School of Computer Science, Carnegie Mellon University.
- Heffernan, N. T. (2002). *Affordable cognitive modeling authoring tools using HCI methods*: Worcester Polytechnic Institute portion. Grant proposal to Office of Naval Research.
- Koedinger, K. R., Anderson, J. R., Hadley, W. H., & Mark, M. A. (1995). Intelligent tutoring goes to school in the big city. *In Proceedings of the 7<sup>th</sup> World Conference on Artificial Intelligence in Education*, 421-428. Charlottesville, VA: Association of the Advancement of Computing in Education.
- Major, N., Ainsworth, S., & Wood D. (1997). REDEEM: Exploiting symbiosis between psychology and authoring environments. *International Journal of Artificial Intelligence in Education*, 8, 317-340.
- Munro, A. (1995). *RIDES QuickStart*. Los Angeles: Behavioral Technology Laboratories, University of Southern California, 1995.

- Munro, A., Johnson, M., Pizzini, Q., Surmon, D., Towne, D., & Wogulis, J. (1997). Authoring simulation-centered tutors with RIDES. *International Journal of Artificial Intelligence in Education*, 8, 284-316.
- Murray, T. (1998). Authoring knowledge-based tutors: Tools for content, instructional strategy, student model, and interface design. *Journal of the Learning Sciences*, 7(1), 5-64.
- Murray, T. (1999). Authoring intelligent tutoring systems: An analysis of the state of the art. *International Journal of Artificial Intelligence in Education*, 10, 98-129.
- Sparks, R., Dooley, S., Meiskey, L., & Blumenthal R. (1999). The LEAP authoring tool: supporting complex courseware authoring through reuse, rapid prototyping, and interactive visualizations. *International Journal of Artificial Intelligence in Education*, 10, 75-97.
- Virvou, M., & Moundridou, M. (2001). Adding an instructor modeling component to the architecture of ITS authoring tools. *International Journal of Artificial Intelligence in Education*, 12, 185-211.
- Wilson, S. J., & Thornton, J. (2001). *Authorware 6*. Florence, KY: OnWord Press.
- Woolf, B. P., & Cunningham, P. A. (1987). Multiple knowledge sources in intelligent teaching systems. *IEEE Expert*, 2(2), 41-54.