Interactive Tutoring System for High School Geometry

Dual Degree Report

Submitted in partial fulfillment of the requirements

for the Degree of

Bachelor of Technology

and

Master of Technology

by

Jayanth Tadinada

06D05016

Supervisors

Prof. Sridhar Iyer

Prof. Anirudha Joshi

Department of Computer Science and Engineering,

Indian Institute of Technology, Bombay

June 2011

Dissertation Approval Certificate

Department of Computer Science and Engineering

Indian Institute of Technology, Bombay

The dissertation entitled submitted by **Jayanth Tadinada** (Roll No: 06D05016) is approved for the award of **Dual Degree (BTech + MTech)** in **Computer Science and Engineering** from **Indian Institute of Technology, Bombay**.

Prof. Sridhar Iyer	**Prof. Anirudha Joshi**
CSE, IIT Bombay	**IDC, IIT Bombay**
Supervisor	**Supervisor**
Prof. Vijay Raisinghani	**Prof. Sahana Murthy**
HoD, NMIMS	**CDEEP, IIT Bombay**
External Examiner	**Internal Examiner**

Prof. Girish Saraph
EE, IIT Bombay
Chairperson

Place: IIT Bombay, Mumbai
Date: 28^{nd} **June, 2011**

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Signature : _____

Name : **Jayanth Tadinada**

Roll No : **06D05016**

Date : _____

Abstract

Most tutoring systems and self learning software are restricted to objective type questions for assessment and evaluation. However, objective type questions are not very effective in assessing the students' ability to solve proof problems in mathematics. The purpose of this project is to develop an interactive proof module which guides high school students while solving proof type problems in mathematics. The content creator enters the problems and the acceptable solutions into the system. The student proves a problem by assembling the statements of a proof from a stack of options. The system compares the proof assembled by the student with the model solutions at each step and gives appropriate hints and feedback.

Acknowledgments

I wish to extend my most sincere thanks to my advisors Prof. Sridhar Iyer and Prof. Anirudha Joshi for providing me with the opportunity of being a part of this project and for their consistent directions. I want to thank Mrs. Suchi Srinivas, Mr. Sridhar Rajagopalan and Ms. Anupriya Gupta of Educational Initiatives Pvt. Ltd. for their suggestions and necessary information regarding the problem statement. I express my gratitude to Prof. Sahana Murthy, Akhil Deshmukh and Kumar Lav for their invaluable help in the project. Finally, I wish to thank the Department of Computer Science and Engineering for providing an environment where I could work effciently.

Contents

List of Figures

Chapter 1

Introduction

With computers and the internet becoming increasingly capable and affordable, use of technology in education has opened up avenues that have the potential to change the educational landscape. The educational software market has flourished in the past few years and a lot of research and development is happening on the use of computers as genuine teaching tools.

Learning from a computer has a lot of advantages compared to traditional classroom model. The learner can learn in a one-on-one setting at anytime of the day at his own pace. Intelligent tutoring systems enable the learner to spend more time learning the subject matter after school hours. Educational software can also be more effective than text books because they can provide a higher level of interactivity and can be motivating factor.

Most of the intelligent tutoring systems rely on objective type questions for evaluation and assessment. While smart design of questions and options can ensure the effectiveness of objective questions in achieving the learning objectives, there are areas where they are not suitable for assessment of student's knowledge and understanding of the concepts. One such area is mathematical proofs.

The goal of this project is to build a tutoring module that guides high school students while solving geometry proof type problems. To achieve this goal, the system should be highly interactive and it should be able to provide hints depending on the student's response and common misconceptions specific to the topic.

For such a system to work well, it is important to make the user experience as rich and intuitive as possible. Otherwise using the system itself will become a non-trivial task for the student which might result in cognitive overload of his working memory. Hence the design of the interfaces is very vital for such a system and educational software in general.

Chapter 2 describes some literature survey on student's misconceptions in Mathematics and discusses on how and why they occur. Chapter 3 details out the problem statement, the scope of the problem and the functional requirements for the project. It also discusses the related work in the field and the approach taken to find the solve the problem.

Chapter 4 describes the workflow of the system and the nature of interaction between each of it's modules. The remainder of the chapter describes the interfaces, design and the functionalities of each module. The details, specifics and extent of implementation are explained in chapter 5. Evaluation, possible extensions of the module and future work are discussed in chapter 6,

Chapter 2

Literature Survey

2.1 Why is mathematics difficult?

If there is one thing that all teachers, students, parents, politicians and economists agree upon, it is that mathematics is a difficult subject. Mathematics continues to be one of the most difficult subjects to teach and to learn at the school level. The reason partly could be because mathematics is presented as a mysterious subject in which definitions, constructs and theorems etc. are to be accepted as they are! This approach to teaching math inevitably creates a socio-mathematical norm in which the student feels that mathematics does not always has to make sense, that the teacher always knows and arbitrates the answer, and that math gets hard as the student progresses in school. [1, chap. 1]

As a result of this approach, a fear of mathematics arises in the minds of the students because they do not have an intuitive understanding of the numbers and the abstract constructs that are built upon them. Without this intuitive understanding, students develop misconceptions as early as primary school which are seldom identified and corrected. These misconceptions are carried forward to higher classes with increasing complexity.

However, it is not fair to entirely blame the teachers or the educational system for the way it is taught because math is indeed hard. Math has its own language, new words, context and arrangement. It is the formal introduction to logic and teaches children how to think and reason. It takes place entirely in the mind and hence students may find it non-tangible and overwhelming sometimes."[1, chap. 1]

2.2 All about misconceptions

2.2.1 What are misconceptions?

Mistakes are an essential part of learning. Mistakes can either be *slips*, *errors* or *misconceptions*. It is important to distinguish between the three. Slips are wrong answers due to processing faults. They are not systematic and are carelessly made by both experts and novices. They can be easily detected and spontaneously corrected.

Errors are wrong answers due to planning; they are systematic in that they are applied regularly in the same circumstances. Errors are symptoms of the underlying conceptual structures that are the cause of the errors. These underlying beliefs and principles in the cognitive structure that are the cause of systematic conceptual errors are called **misconceptions**. [12]

2.2.2 Learning theories

Two opposing learning theories are outlined below in a simple manner that will illustrate different approaches to handling pupils' misconceptions. [8]

Behaviorist theory

The behaviorist theory of learning assumes that knowledge has a separate universal existence and learning is defined as transfering that knowledge into the mind. The pupil is viewed as a passive recipient of knowledge, an "empty vessel" to be filled, a blank sheet (*tabula rasa*) on which the teacher can write. This theory sees learning as conditioning the correct responses through stimulus-response bonds. These stimulus-response bonds are strengthened by success and reward (positive reinforcements) and weakened by failure and punishments (negative reinforcements).

From a behaviorist perspective, errors and misconceptions are not important because, it does not consider pupils' current concepts as relevant to learning. Errors and misconceptions are rather seen as faulty bytes in a computer's memory *i.e.* those which can be erased and overwritten through drill and practice.

Constructivist theory

The constructivist theory of learning assumes that concepts are not directly taken from experience, but built upon existing knowledge. Hence a student's ability to learn from an experience depends on the quality of his observations and the quality of his pre-existing theories. The student therefore is not seen as a passive recipient of knowledge but as an active participant in the construction of his own knowledge. This construction activity involves the interaction of a child's existing ideas and new ideas. New ideas are interpreted and understood in the light of that child's current knowledge. In addition to interpretation, children also organize and structure this knowledge into large units of inter-related concepts called **schemas**.

When a new idea is presented to the student, the idea is interpreted in terms of an existing schema. This is called **assimilation**. Sometimes the new idea may be quite different from his existing schemas; in which case the student reconstructs and reorganizes his existing schema to accommodate the new idea with the previous knowledge intact. This is called **accommodation**.

From a constructivist perspective, misconceptions occur because of the gaps between existing schemas and the new ideas presented.

2.2.3 Why do students get misconceptions?

Rote learning

If a new idea is so different from the student's existing schemas that assimilation or accommodation is impossible, then the student creates a "box" that is entirely disconnected with the existing knowledge schemas and tries to memorize the idea. This is called *rote learning*. In such a case, the idea is not understood because it is not linked to any existing knowledge. This isolated knowledge is also difficult to remember. Such rote learning is the cause of many mistakes and misconceptions among students. [8]

Intelligent overgeneralization

One of the main sources of misconceptions is overgeneralization of previous knowledge (that was correct in an earlier domain), to an extended domain (where it is not

valid). For example, assumption of commutativity in subtraction is a common misconception. The children first observe commutativity in addition and multiplication when learning their tables. *In lieu* of any contradictory evidence, they have no reason to expect that subtraction will behave otherwise. In this case, the children are just overgeneralizing over operators. Similar overgeneralization over operators has been observed in division, and also in algebra. [14]

Sticking to misconceptions

"Erroneous conceptions are so stable because they are not always incorrect. A conception that fails all the time cannot persist. It is because there is a local consistency and a local efficiency in a limited area, that those incorrect conceptions have stability".

For example, a child newly learning about decimal system might overgeneralize his prior knowledge of integers and think that $5.25 > 5.5$ since $25 > 5$. The problem with this flawed conceptual model is that it works in a lot of cases when both the numbers compared have the same number of digits after the decimal point. [12]

Conclusion

From a constructivist perspective, errors and misconceptions are a natural result of children's effort to construct their own knowledge. These misconceptions should not be looked down as terrible things that need to be eliminated because that might shake the child's confidence on his previous knowledge which will only lead to more misconceptions. These errors should be regarded as a part of the learning process and as an opportunity to enhance learning.

2.3 A few common misconceptions

2.3.1 Counting numbers and early arithmetic

Misconceptions in mathematics can spring right from primary school. It is important to identify them early on and resolve them. It is also important to study and understand why misconceptions occur so that teaching strategies could be better designed to avoid the misconceptions.

Counting troubles diagnose early problems

Counting is a very culture specific activity. Children who learn to count in English often find it hard to make relations and see patterns in numbers. Japanese children are found to have an advantage in recognizing patterns in numbers faster than their English counterparts because linguistically, Japanese numbers capture the pattern more efficiently than English which has an irregular sequence around '*eleven, twelve, thirteen, fourteen and fifteen*'. Research has shown that this linguistic failure of the English language explains many children's delay in the mastery of the sequence of numerals needed for counting. However, this advantage is only shortlived for the Japanese students as most students make up for the initial hiccups irrespective of the language. [14, chap. 4]

Subtracting *smaller* from *larger* over-generalization

When children are initially taught subtraction, they are usually taught 'simple subtraction without decomposition' first followed by two digit subtraction that involves carrying. Teachers while teaching simple subtraction often give thumb rules like "*always subtract smaller number from larger*" which the student practices and masters. But this rule of thumb is not valid in case of two digit subtraction with carrying. However, students intuitively tend to over-generalize rules like these and end up making mistakes like $32 - 17 = 25$. [14, chap. 4]

2.3.2 From positive numbers to negative numbers

As we shall see below, a number of intelligent over-generalizations that the students make when dealing with positive numbers break down when encountered with integers. This intelligent over-generalization is the root cause for quite a few misconceptions. However, most of these misconceptions can be cleared with the use of a number line. [14, chap. 3]

Why is *minus* times *minus plus*?

The negative numbers present learners with many problems when they are first introduced and continue to be a problem for children who think mathematics ought to be sensible. When the child for the first time sees something like *subtracting* $-$

$1\ from + 5\ gives + 6$, she wonders how can one take away something from a number and end up with a bigger number!

These are not easy questions to answer and unfortuntely, enough time is not devoted by the teachers to address doubts like these. Perhaps because the student fails to get answers that satisfy his intuition and common sense, she begins to think that negative numbers are eccentric and capricious. [14, p. 111]

Other intelligent over-generalizations that fail

Children make a lot of generalizations when they are learning subtraction, multiplication etc. Most of these generalizations are true over Whole numbers and are also easily verifiable by intuition. Also most of these generalizations fail when the child over-generalizes the same thumb-rules over integers.

An important example of such an over-generalization is the belief that '*multiplication makes bigger*' (or that '*division makes smaller*'). This is true in most cases over Whole numbers but fails in case of integers and rational numbers. So children end up making mistakes like $0.3 \times 0.2 = 0.6$ or that $0.5^2 > 0.5$ etc. [16]

2.3.3 Fractions and rational numbers

Modeling fractions – the fallacies of the cake model

Studies have shown that using the cake model does not mean that the student grasps the concept of ratio and proportion. In a study, only 7% of 8-year old students have correctly recognized that $\frac{1}{2}$, $\frac{2}{4}$ and $\frac{4}{8}$ are the same shaded fraction of a circle.

Most of the children who have grasped the concept of part-whole relationship with respect to the cake model may be completely baffled when the whole rectangle is cut into parts in different ways. In fact 75% of children aged nine have agreed that if a round cake is divided into four parts using four parallel lines, then each piece is one-fourth of the cake.[14, chap. 4]

Division and multiplication of rational numbers

Students often have a common misconception that decimals and fractions are different types of numbers and hence find the concept of finding equivalent fractions confusing and counterintuitive. Their intuition further breaks down when they have to divide a whole number by a fraction. The concept of dividing by 2 or 5 is intuitive for them

but division by $\frac{1}{4}$ is not straightforward and requires more thought. This difficulty can be overcome by providing the student with appropriate scaffodling. For example, the children can be taught to think of $3 \div \frac{1}{4}$ as 'the number of $\frac{1}{4}$'s that fit into 3'.

Multipying decimals is also an area where students tend to have a lot of misconceptions. Students are quick to generalize from $1 \times 1 = 1$ that $0.1 \times 0.1 = 0.1$. Also errors like $2.3 \times 10 = 2.30$ or 20.3 etc. are quite common. [16]

2.3.4 Geometry and measurement

Identifying known geometric objects in a different orientation

Students are heavily influenced by the prototypes that they have been presented with. For instance, in a typical math textbook, all rectangles are shown to be lying 'flat' on one of its longer sides. This often leads to a situation where the children fails to recognize a rectangle when it is in a different orientation. Students also have trouble recognizing that a square is also a rectangle and that a rectangle is also a parallelogram. [14, chap. 4]

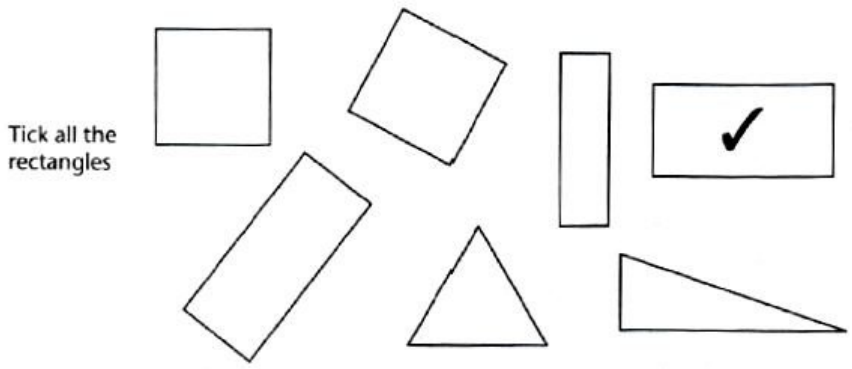


Figure 2.1: In a task given to fourth graders, a significant number of students answered this way [14, p. 20]

From angles to triangles

Children are taught to identify acute, obtuse, right and straight angles. And soon they are introduced to the concept of acute angled, right angled and obtuse angled triangle. In a task given to sixth graders, more than 50% of the students were unable

to explain why a triangle is obtuse angled even though the triangle has two acute angles and only one obtuse angle. [6]

2.3.5 Ambiguity in the mathematical language

It is important for the student to grasp and understand the terminology, what it means in a mathematical sense and realize the incoherence between the real world meanings of the word with the mathematical usage of them. "In the real world, the student has little problem realizing that a town square is not really square and that the edge of a stream is not really straight but it is not so in the case of mathematics". [14, chap. 5] Here it is important for the student to understand the mathematical sense when words like 'square' and 'edge' are used. Failure to grasp this mathematical language leads to mistakes like these:
Question: *How many times can you subtract 7 from 83, and what is left afterwards?*
Response from a student: *I can subtract as many times as I want, and it gives 76 every time.* [14, chap. 5]

2.4 Some computer based teaching tools

Although a number of applications, softwares and interactive tools were developed in the recent years to help teach mathematics, most of them are focused on students of ages 4 - 9. It is important to notice that as the target age of the students increases, the degree of interactivity in the system decreases and tend to take a standard, textbook style approach without much interactivity. We review a few tools that are relevant to the nature of the project in this section.

2.4.1 Mindspark

Mindspark is an adaptive self learning programme that helps children improve their skills and conceptual understanding of the subjects in their curriculum. The student interactively answers questions of progressively increasing complexity and learning happens through answering the questions. The questions are carefully designed to contain interactive animations and explanations which aim at strengthening the conceptual understanding of the subject. Extremely finely graded questions allow the

system to analyze a student's understanding and pin point misconceptions.

The system has a built in **adaptive logic** that allows the student to learn at her own pace. Typically, a student would select a topic, and start with fundamental questions based on the topic. A student with a deeper understanding of the topic automatically moves ahead quickly to more advanced levels, while those who need to spend more time on the basics will be served up more questions of the basic level.

Whenever a student goes wrong in a particular type of question, a detailed explanation is given addressing the conceptual misunderstanding and common misconceptions. These explanations are usually **visual or animated** that engage students and increase their motivation levels. If a student does poorly in a particular topic, she is asked to take a remedial item which is much more interactive and focuses on the basics.

Each student has a personal account in which her performance is tracked. Students are awarded animated light bulbs called "sparkies" if they do well consistently thereby providing them with some playful motivation. The system also provides **live feedback** to parents and teachers so that they can track the child's progress. For a parent, the report provides details of how the child is doing in different topics and concepts. For a teacher, the report not only provides a summary of the progress made by her class, but also details of how each student is progressing in different topics and concepts. [11]

2.4.2 Mindspark's Proofs Module

Mindspark has experimented with a simple model for assessing geometry proofs. A screenshot of the model is shown in figure 3.1. Although, the model forces the student to think trough the proof before filling the blanks, it is very rudimentary. It lacks interactivity, scalability, intelligent feedback and more importantly, it gives away all the crucial steps of the solution that require conceptual understanding of the topic. For all aforementioned reasons, this module needs to be completely redesigned from scratch.

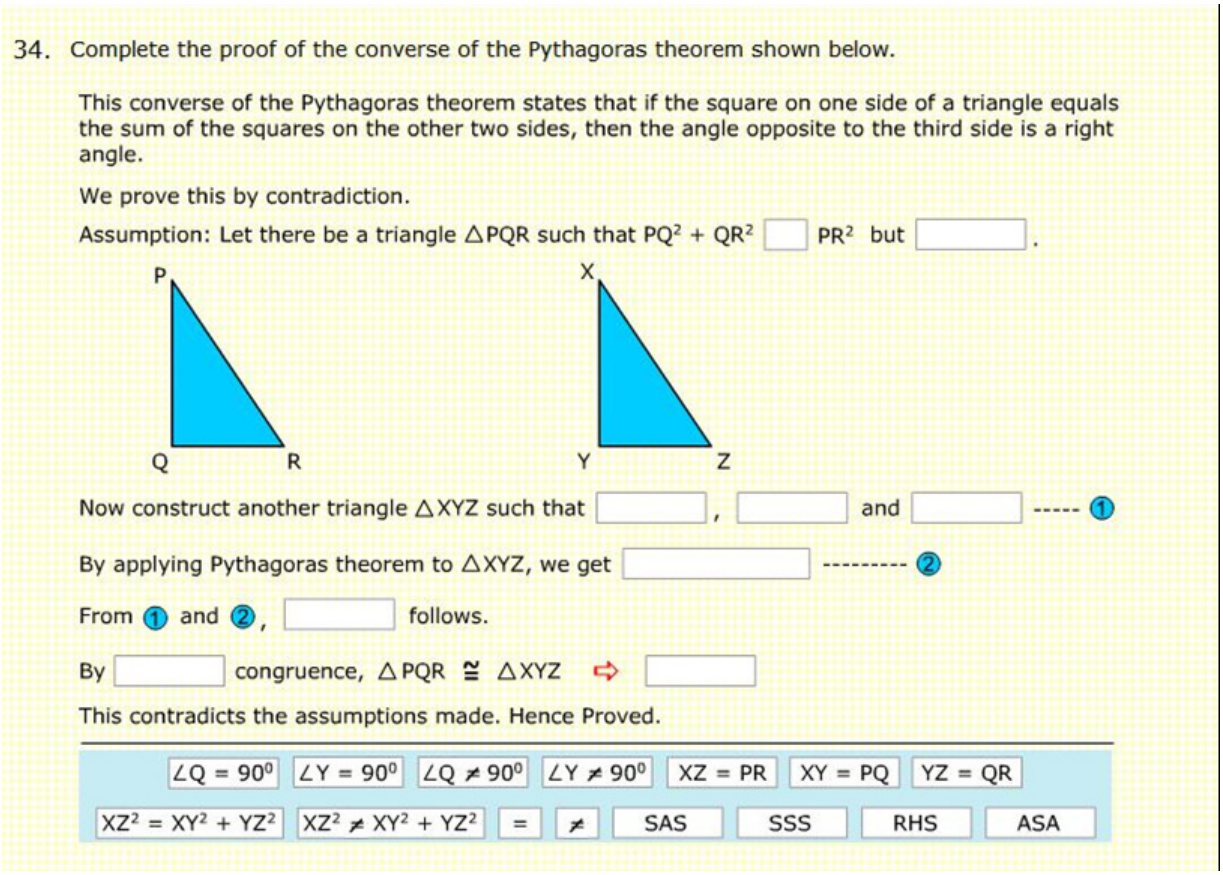


Figure 2.2: A screenshot of Mindspark's existing module for geometry proof type question

2.4.3 Carnegie Learning's Cognitive Tutor

Carnegie Learning, Inc. is a publisher and distributor of innovative research-based mathematics curricula for middle-school, high-school and post-secondary students. Cognitive Tutor is an intelligent tutoring software developed by Carnegie Learning that is based on J. Anderson's ACT* theory of learning.

According to ACT*, all knowledge begins as declarative information; procedural knowledge is learned by making inferences from already existing factual knowledge. ACT* supports three fundamental types of learning: generalization, in which productions become broader in their range of application, discrimination, in which productions become narrow in their range of application, and strengthening, in which some productions are applied more often. New productions are formed by the conjunction or disjunction of existing productions. [7]

Figure 3.2 shows a screenshot of a geometry proof problem in cognitive tutor. The question is displayed on the left side of the screen and the student has to build a flowchart for the proof on the right side of the screen using the interface. Each box in the flowchart represents a step in the proof and each arrow represents implication. The system gives live feedback and intelligent hints to guide the student through the solution and also assesses the students performance and identifies his/her misconceptions in the topic.

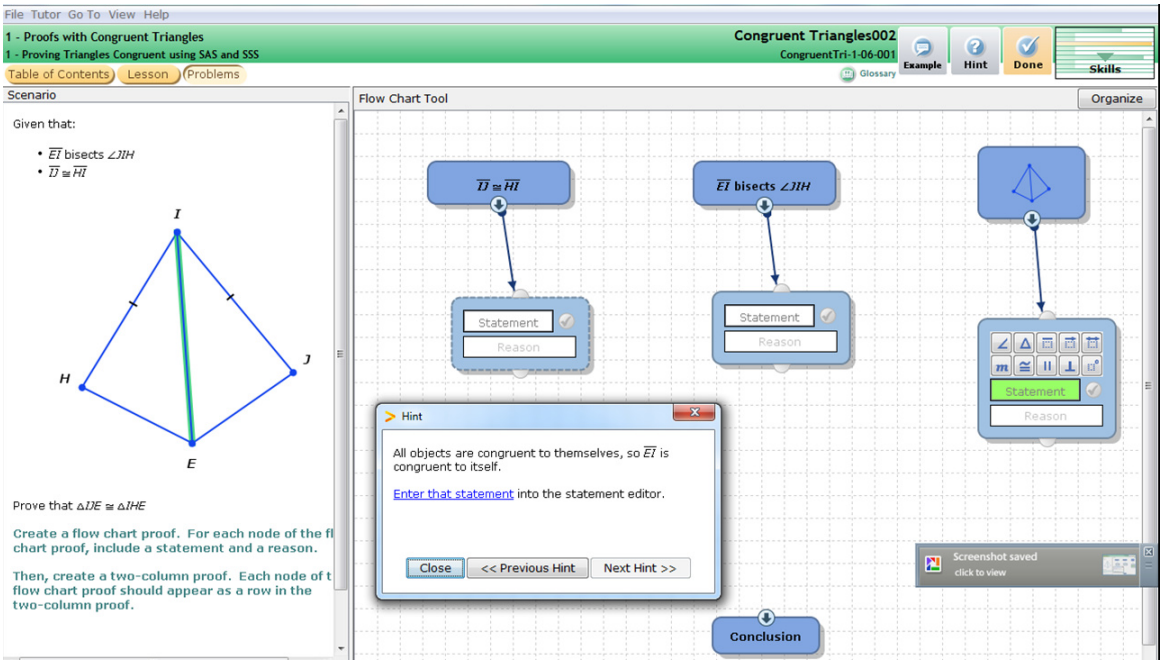


Figure 2.3: A screenshot of Carnegie Learning's Cognitive Tutor

The cognitive tutor is now implemented in a few of counties in the US. A number of studies conducted on control groups have shown that a majority of teachers and students have reacted positively to cognitive tutor showing improvements in both efficiency and test scores. But a study of 10 schools that implemented Cognitive Tutoring by The Department of Educational Accountability, Fairfax County, Virginia showed that students who had cognitive tutor did not outperform their traditional mathematics counterparts in Standards of Learning (SOL) Algebra tests across two and a half years. On the SOL Geometry total test, only in one year, students in cognitive tutor schools demonstrated better performance than their tradition mathematics curriculum counterparts. [18]

2.4.4 Other Commercial Tools and Packages

A number of commercial tutorial and self help packages like Math Success Deluxe, Math Advantage, High Achiever etc. are available online. Most of them are designed specifically for the American curriculum. Their application and utility (by design) is limited to being additional material for homework help rather than as teaching tools for conceptual understanding. Most of the questions at the high school level are numerical in nature in an objective format. The proof type problems are covered but only as a guide with printed solutions at the back and the practice problems lacked any sort of interactivity. [17]

Chapter 3

The Problem Statement

Carefully designed objective type questions can test a student's conceptual understanding of most topics in mathematics at highschool level. Currently all questions in the Mindspark program (with the exception of a few remedial items) are objective type questions. However, there are a few areas where objective type questions are not sufficient. Geometry proofs is one such area where student's ability to construct a proof by applying various rules and theorems cannot be assessed with only objective type questions. The goal of this project is to design and build an interactive geometry proof module for Mindspark which will guide students while solving geometry proof type problems by giving appropriate feedback and hints.

This chapter defines the scope of the project and the functionalities of the proof module, proposes a workflow for the system and discusses a few approaches for solving the problem including their mertis and demerits.

3.1 Existing Work

Although a number of applications, softwares and interactive tools were developed in the recent years to help teach mathematics, most of them are focused on students of ages 4 - 9. For most tools focused on high school students, the explanation and the review problems tend to become more standard and textbook-like as the complexity of the concepts increases.

3.1.1 Mindspark's Proofs Module

Mindspark has experimented with a simple model for assessing geometry proofs. A screenshot of the model is shown in figure 3.1. Although, the model forces the student to think trough the proof before filling the blanks, it is very rudimentary. It lacks interactivity, scalability, intelligent feedback and more importantly, it gives away all the crucial steps of the solution that require conceptual understanding of the topic. For all aforementioned reasons, this module needs to be completely redesigned from scratch.

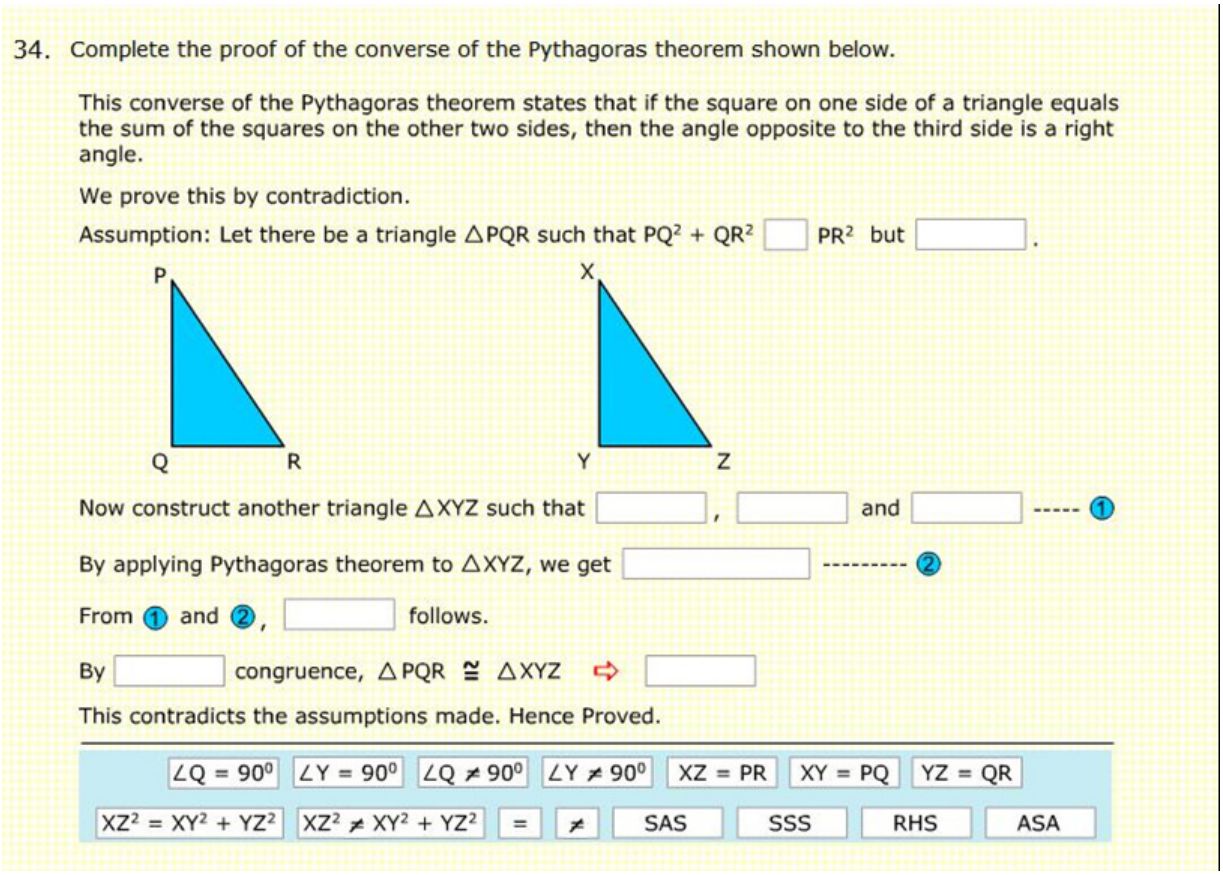


Figure 3.1: A screenshot of Mindspark's existing module for geometry proof type question

3.1.2 Carnegie Learning's Cognitive Tutor

Carnegie Learning, Inc. is a publisher and distributor of innovative research-based mathematics curricula for middle-school, high-school and post-secondary students.

Cognitive Tutor is an intelligent tutoring software developed by Carnegie Learning that is based on J. Anderson's ACT* theory of learning.

According to ACT*, all knowledge begins as declarative information; procedural knowledge is learned by making inferences from already existing factual knowledge. ACT* supports three fundamental types of learning: generalization, in which productions become broader in their range of application, discrimination, in which productions become narrow in their range of application, and strengthening, in which some productions are applied more often. New productions are formed by the conjunction or disjunction of existing productions. [7]

Figure 3.2 shows a screenshot of a geometry proof problem in cognitive tutor. The question is displayed on the left side of the screen and the student has to build a flowchart for the proof on the right side of the screen using the interface. Each box in the flowchart represents a step in the proof and each arrow represents implication. The system gives live feedback and intelligent hints to guide the student through the solution and also assesses the students performance and identifies his/her misconceptions in the topic.

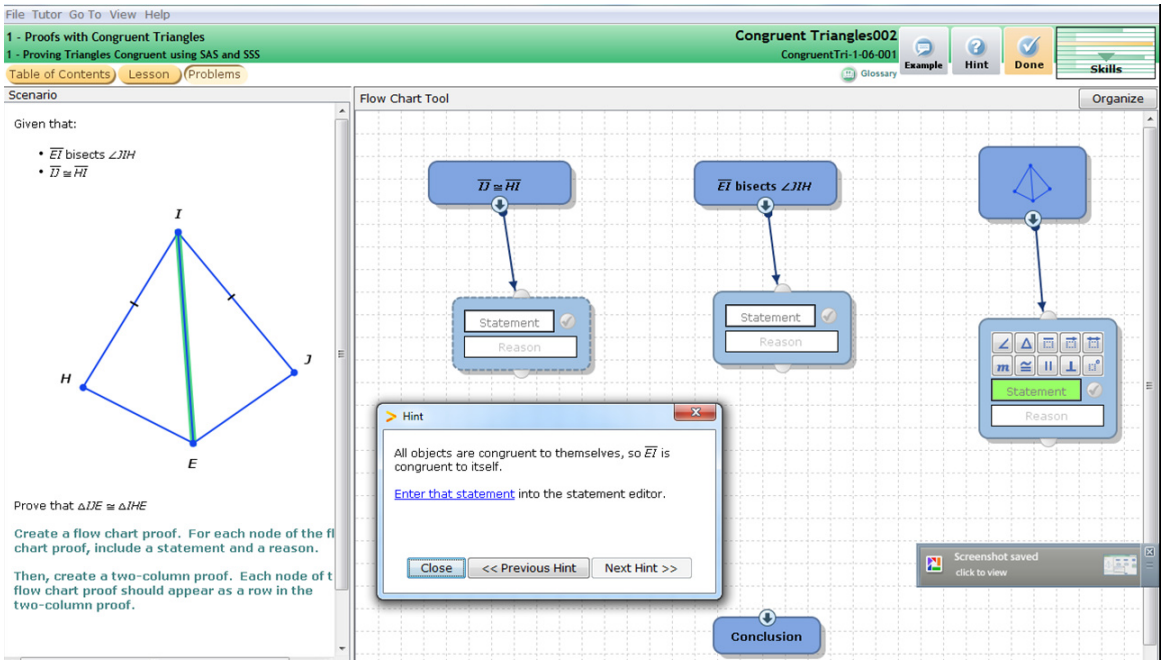


Figure 3.2: A screenshot of Carnegie Learning's Cognitive Tutor

The cognitive tutor is now implemented in a few of counties in the US. A number of studies conducted on control groups have shown that a majority of teachers and students have reacted positively to cognitive tutor showing improvements in both efficiency and test scores. But a study of 10 schools that implemented Cognitive Tutoring by The Department of Educational Accountability, Fairfax County, Virginia showed that students who had cognitive tutor did not outperform their traditional mathematics counterparts in Standards of Learning (SOL) Algebra tests across two and a half years. On the SOL Geometry total test, only in one year, students in cognitive tutor schools demonstrated better performance than their tradition mathematics curriculum counterparts. [18]

3.1.3 Other Commercial Tools and Packages

A number of commercial tutorial and self help packages like Math Success Deluxe, Math Advantage, High Achiever etc. are available online. Most of them are designed specifically for the American curriculum. Their application and utility (by design) is limited to being additional material for homework help rather than as teaching tools for conceptual understanding. Most of the questions at the high school level are numerical in nature in an objective format. The proof type problems are covered but only as a guide with printed solutions at the back and the practice problems lacked any sort of interactivity. [17]

3.2 Scope and Functional Requirements

The project aims to make a tool which helps students solve geometry proof problems specific to congruency and similarity of triangles. The design of the interface and the architecture should be flexible enough to be extended to proof type problems in all topics of mathematics. An initial list of functional requirements has been prepared after meetings with various members of the Mindspark team. A description of the system is given through the initial functional requirements below:

- An intuitive interface for the content creator to upload questions, figures and model solutions into the database.
- On the student end, a question is retrieved from the database and presented to the student along with the images (if any)

- The student should be able to make geometrical constructions that are required for solving the problem (dropping a perpendicular from a point onto a line etc.)
- The system should have constraints that force the students to explicitly state what is given in the problem and what is to be proved. This is in accordance with the NCERT guidelines for evaluating proof type problems.
- The students will simply type the proof. The interface for typing down mathematical symbols and equations should be child friendly to minimize cognitive overload of the students' working memory.
- The proof consists of a series of statements. Each statement, however trivial should be supported by a reason. The system should check the validity of the statement and give appropriate feedback.
- While typing the statements and reasons, an auto-complete suggestion mechanism is desirable. (optional requirement)
- The reason for every statement will be selected from a drop down list. Elements in the drop down list should be automatically populated by the system.
- The student should be able to mark equations and be able to refer them in the subsequent steps of the proof.
- Whenever the student makes a mistake, the system should suggest intelligent hints which will guide the student through the step. The hints should be based on student response and common misconceptions in that particular topic.
- There should be a "hint" button in the interface which the student can use at any point of time.
- There should be a "show next step" button to show the next step to the student.
- The module should seamlessly integrate into the existing Mindspark system.

3.3 Approach

Given the scope of the problem, the system should be designed in such a way that it is easily extendable to proof type problems in other areas of mathematics and also

leave enough scope for improving individial components of the system like hints, user interface etc. To allow the flexibility, the system is designed in a modular fashion. Figure 3.3 shows the work flow of the system.

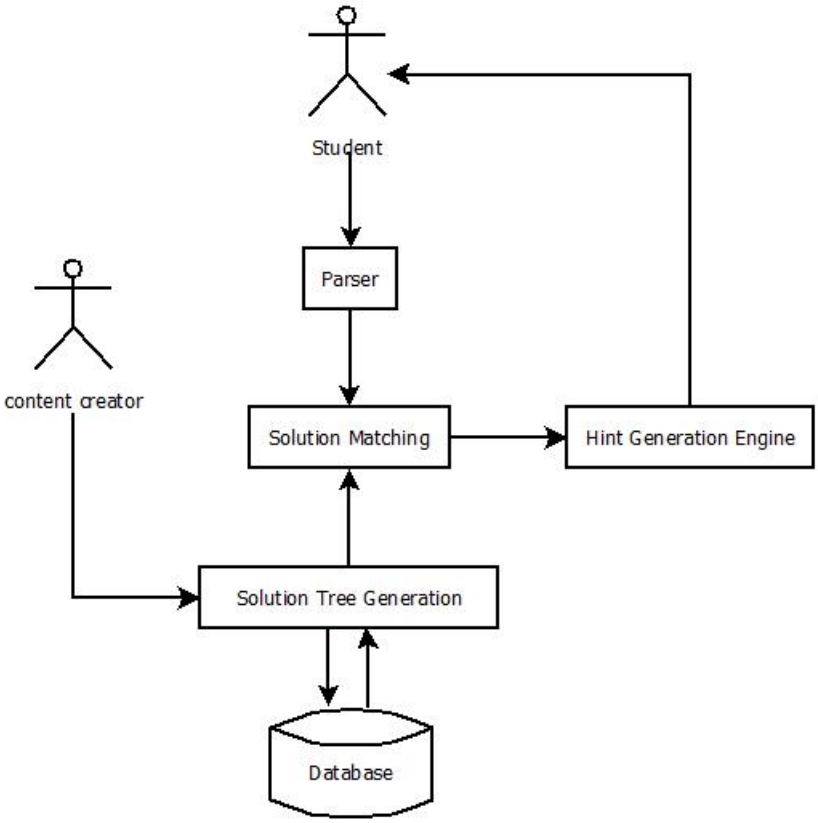


Figure 3.3: Workflow of the system showing all the modules

The crux of the design lies in the way the solution tree is modeled. Two approaches that were considered (but later found inadequate) to model the solution tree are discussed below.

3.3.1 The Tree Model

To explain this approach, we use the following example problem.
Example 1: *In* $\triangle ABC$, *BD and CE are perpendiculars to sides AC and AB of with* $BD = CE$. *Prove that ABC is an Isosceles triangle.*

There are numerous ways to solve this problem using only the properties of triangles. Let us consider four of the possible solutions. Each statement in the proof

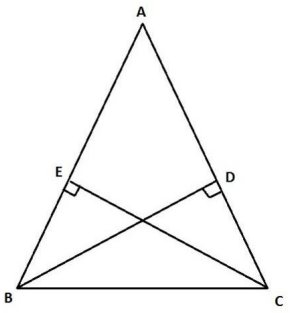


Figure 3.4: Figure for Example 1

is identified by a unique number. If the same statement appears in two different solutions, the same number is used to identify that statement.

Solution 1 (SOL1): This is the most standard and expected solution for the problem where we use the congruency property of triangles.
Given:
1. $\angle AEC = 90$
2. $\angle BDA = 90$
3. $BD = CE$
To Prove:
6. $AB = AC$
Proof:
In $\triangle ABD$ and $\triangle ACE$
3. $BD = CE$ (given)
2. $\angle AEC = 90 = \angle BDA$ (given)
4. $\angle A = \angle A$ (common angle)
5. Therefore, $\triangle ABD \cong \triangle ACE$ (by A.A.S property)
6. $AB = AC$ (corresponding parts of congruent triangles)

Hence Proved

Solution 2 (SOL2): SOL2, as we shall see is similar to SOL1 except that A.S.A property is used to prove the congruency between $\triangle ABD$& $\triangle ACE$ instead of A.A.S property that is used in SOL1.
Given:

1. $\angle AEC = 90$
2. $\angle BDA = 90$
3. $BD = CE$

To Prove:

6. $AB = AC$

Proof:

In $\triangle ABD$ and $\triangle ACE$

3. $BD = CE$ (given)
7. $\angle ABD = 90 - \angle A$ (from figure)
8. $\angle ACE = 90 - \angle A$ (from figure)
9. $\angle ABD = \angle ACE$ (from 7. and 8.)
5. Therefore, $\triangle ABD \cong \triangle ACE$ (by A.S.A property)
6. $AB = AC$ (corresponding parts of congruent triangles)

Hence Proved

Solution 3 (SOL3): The proof in this case is a little different from the above two solutions. In order to prove that $\triangle ABC$ is isosceles, we prove that $\angle ABC = \angle ACB$ instead of proving $AB = AC$ as in the above solutions.

Given:

11. $\angle BEC = 90$
12. $\angle BDC = 90$
3. $BD = CE$

To Prove:

13. $\angle ABC = \angle ACB$

Proof:

In $\triangle BDC$ and $\triangle BEC$

3. $BD = CE$ (given)
11. $\angle BEC = 90 = \angle BDC$ (given)
14. $BC = BC$ (common side)
15. Therefore, $\triangle BDC \cong \triangle BEC$ (by R.H.S property)
16. $\angle EBC = \angle DCB$ (corresponding parts of congruent triangles)
13. $\angle ABC = \angle ACB$ (Same angle as above)

Hence Proved

Solution 4 (SOL4): This proof is entirely different from the above three proofs. Here, we take a more algebraic approach to solving the problem rather than a geometry approach. We calculate the area of the triangle in two different ways and equate the areas to prove that $AB = AC$

Given:

1. $\angle AEC = 90$
2. $\angle BDA = 90$
3. $BD = CE$

To Prove:

6. $AB = AC$

Proof:

17. $Area\ of\ \triangle ABC = \frac{1}{2} \cdot BD \cdot AC$ (computing area of triangle)
18. $Area\ of\ \triangle ABC = \frac{1}{2} \cdot CE \cdot AB$ (computing area of triangle)
19. $\frac{1}{2} \cdot BD \cdot AC = \frac{1}{2} \cdot CE \cdot AB$ (equating areas)
19. $BD \cdot AC = CE \cdot AB$ (algebra manipulation)
6. $AB = AC$ (because $BD = CE$)

Hence Proved

In the Tree Model for solution matching, all the correct solutions are represented by a Solution Tree. Each state consists of the set of all valid statements entered by the user at that stage. The hypothesis (*i.e.* the set of statements that are *given*) becomes the root of the Solution Tree. When a valid statement along with the correct explanation is entered by the user, a transition is made in the tree depending on the input. The figure below illustrates how the transitions are made. All the numbers shown in each state correspond to the statements in SOL1, SOL2, SOL3 and SOL4. The Solution Tree with four solutions for the example question that was discussed above is represented in fig 3.5.

SOL1 and SOL2 are very similar except for the fact that they use two different rules to prove the same thing and yet in the tree model, they are two completely different branches separated at the root. SOL1 and SOL3 interpret the same hypothesis in slightly different ways but in this model SOL1 and SOL3 are completely disjoint

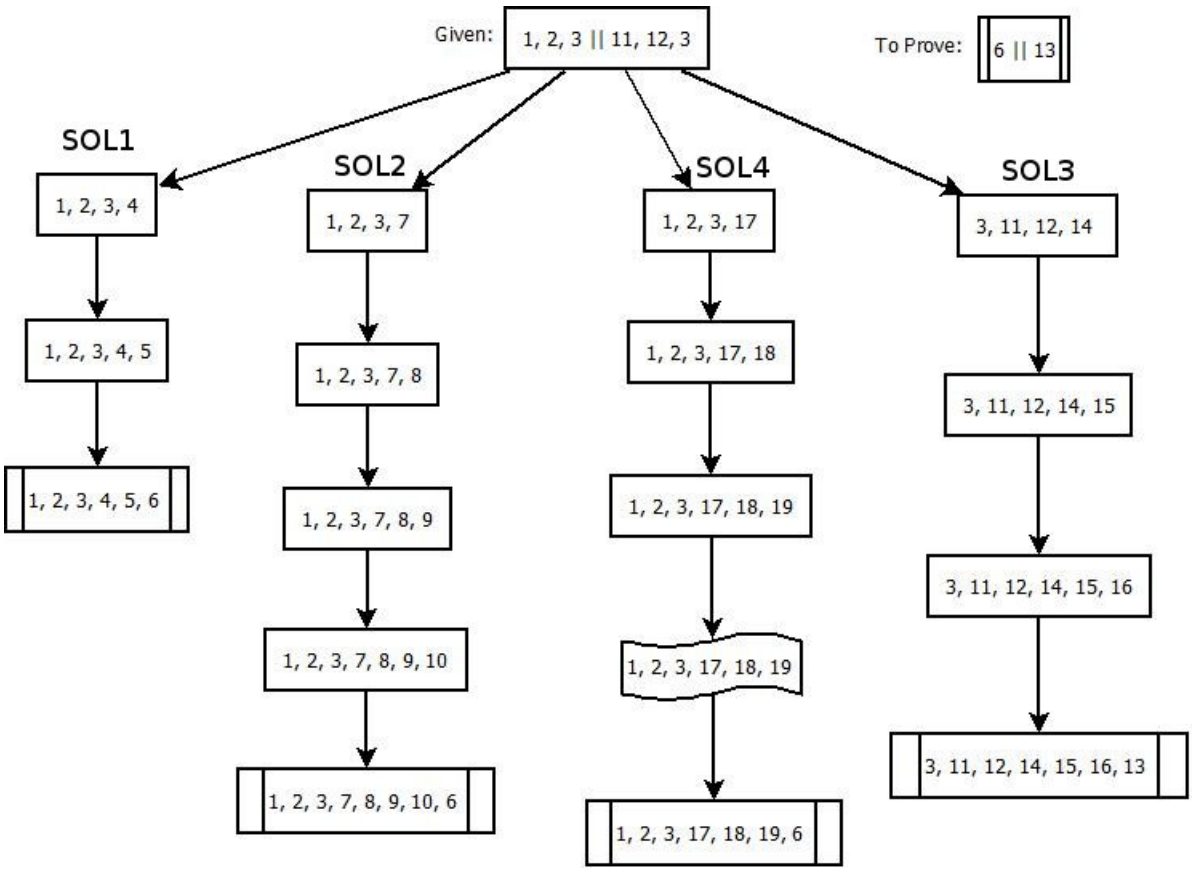


Figure 3.5: The Tree representation of the four solutions of Example 1

for all practical purposes.

In Solution 4, the transition from state [1, 2, 3, 17, 18, 19] $\rightarrow$ {1, 2, 3, 17, 18, 19} shows that algebraic manipulations using this models is inefficient in space. Also, changing the order of the steps in any solution will spawn a completely different branch in the solution tree. Thus, this model is very inefficient in both space and time and hence we discard this model.

3.3.2 The Box Model

To illustrate this model, we use Example 2 which is a modification of Example 1 that is used to describe the tree model.

Example 2: *ABC is an Isosceles triangle with $AB = AC$. BD and CE are perpendiculars dropped on AC and AB respectively. Prove that $BD = CE$.*

Proof 1 (P1): is the straight forward and expected solution to the problem using

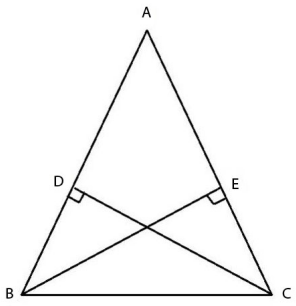


Figure 3.6: Figure for Example 2

the congruency of triangles

Given :

1. $AB = AC$
2. $\angle BDC = 90$
3. $\angle BEC = 90$

$ToProve$:

9. $BE = CD$

In $\triangle ABE$ and $\triangle ACD$

4. $\angle A = \angle A$ (common angle)
1. $AB = AC$ (given)
5. $\angle ABE = 90 - \angle A$ (from figure)
6. $\angle ACD = 90 - \angle A$ (from figure)
7. $\angle ABE = \angle ACD$ (from 6. and 7.)
8. $\triangle ABE \cong \triangle ACD$ (by A.S.A property)
9. $BE = CD$ (corresponding parts of congruent triangles)

Hence Proved

Proof 2 (P2): is similar to P1 except that a different pair of triangles are proved to be congruent to arrive at the same result.

Given :

10. $\angle ABC = \angle ACB$
2. $\angle BDC = 90$
3. $\angle BEC = 90$

$ToProve$:
9. $BE = CD$
In $\triangle BDC$ and $\triangle CEB$
11. $BC = BC$ (common side)
5. $\angle ABE = 90 - \angle A$ (from figure)
6. $\angle ACD = 90 - \angle A$ (from figure)
7. $\angle ABE = \angle ACD$ (from 6. and 7.)
12. $\angle EBC = \angle ABC - \angle ABE$ (from figure)
13. $\angle DCB = \angle ACB - \angle ACD$ (from figure)
14. $\angle EBC = \angle DCB$ (from 10, 7, 12 and 13.)
10. $\angle ABC = \angle ACB$ (given)
15. $\triangle BDC \cong \triangle CEB$ (by A.S.A property)
9. $BE = CD$ (corresponding parts of congruent triangles)

Hence Proved

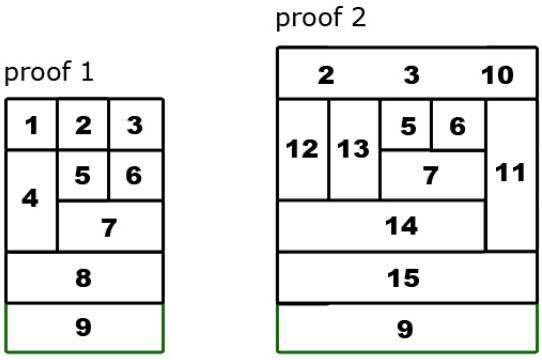


Figure 3.7: The Box representation of the two solutions of Example 2

The Box Model for solution matching, is built using the predicate logic principles for each layer of the box. All the steps at the same level are mutually independent and the order of entering them in the solution does not matter. A step in a lower level can be entered only when all the steps in the level above it are entered. For example, *step 5* and *step 6* can be entered in any order whereas *step 7* can be entered

only when both *step 5* and *step 6* are already entered. Figure 3.7 contains the Box Models for Proofs P1 and P2. The numbers shown in the boxes correspond to the statement numbers in P1 and P2.

Unlike the Tree Model, the user can enter statements in any order as long as it is valid *i.e.* and the Box model handles it without using any extra space. The disadvantages of this model is that the Boxes are not easy to build and the content creator has to create models for every possible solution which in itself is a very tedious task. The problem of representing algebraic manipulation is not addressed in this model too. Though it is better than the Tree Model, the Box Model is also inefficient and very tedious to implement and use.

Given the constraints of memory and processing power available in a browser and the fact there are a lot of issues on interface design and hint generation which need focus, the problem has to be scoped down to a simpler version of the problem. The modified problem statement and the final design of the system is discussed in the next chapter.

Chapter 4

Design

In the new scoped down model for geometry proofs module, the student instead of manually typing each step of the solution has to assemble the proof given a set of possible steps using a drag-and-drop mechanism on the interface. This is the first step towards achieving the higher goal that is to enable the student to type the proof into the system in a way that is as intuitive as writing it down on paper.

The work flow of the system is shown in figure 4.1. The system has five modules – The Proof Assembly Module (PA), The Content Creation (CC) module, The Solution Tree (ST) module, The Solution Matching (SM) module and The Hint Generation (HG) Module. The PA module is essentially the student's interface using which the proof is assembled. The CC module is for the content creator to enter questions and all acceptable solutions into the system. The solutions are represented using an XML Tree model. Once the solutions are entered, they are merged to form a General Solution Tree (GST) in the ST module which will be used by the SM module for solution matching. If the student makes any mistake or requests for a hint, then the HG module is invoked and approriate hints are suggested. The details and design of each module is described in this chapter.

4.1 Proof Assembly Module

The PA module is essentially the student's interface for the system. A wire frame of the PA module is shown in fig. 4.2.

The interface consists of three major divisions the functionalities of which are discussed below:

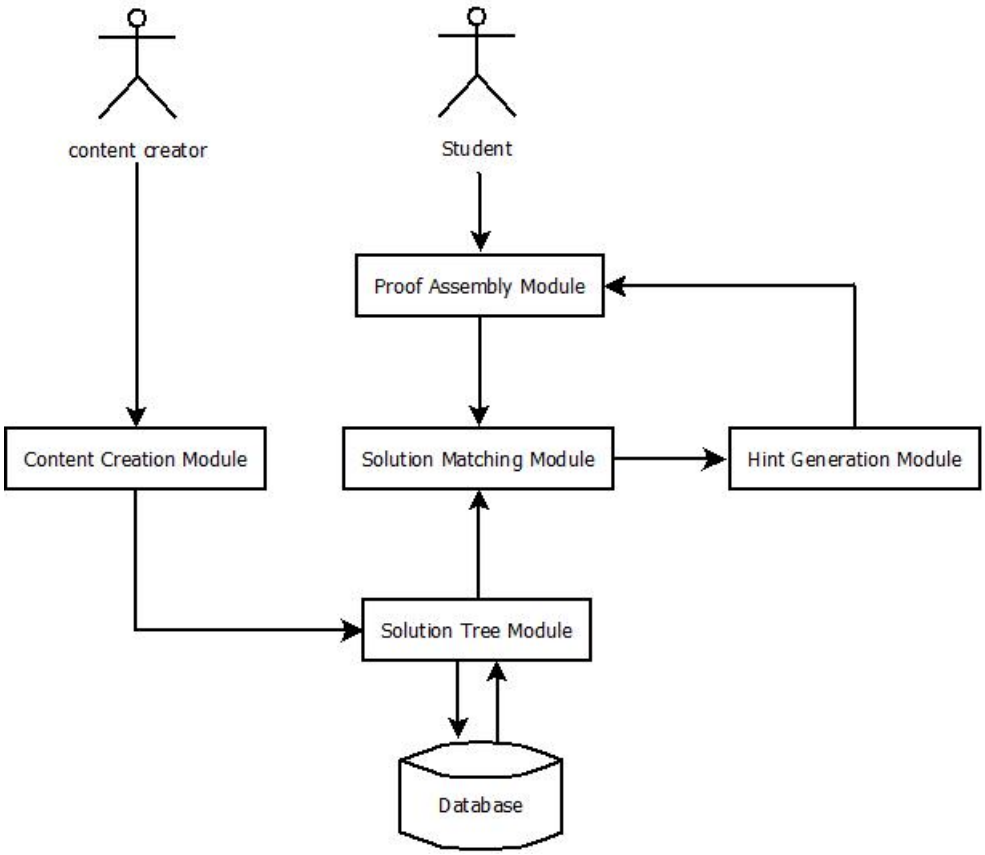


Figure 4.1: Workflow of the system showing all the modules

- **The Question:** The question is displayed in this division. If the question contains an image, then it is also displayed. Sometimes, the student will have to do constructions (like dropping a perpendicular etc.) while solving the proof. In such cases, the image initially displayed will be replaced by a new image.

- **The Assertion Stack:** On the right is the assertion stack. A fixed number of assertions are displayed in the stack at any point of time. The student can drag and drop the assertions into the proof assembly area. The assertion stack refreshes itself to give new options after every step.

- **Proof Assembly:** To complete the solution, the student has to pick correct assertions one by one and place them in the correct order. Every assertion should be justified by a reason. The reason is chosen from a drop down menu that is automatically populated by the system whenever an assertion is made.

  There are options in the interface for the student to ask for a hint or the next

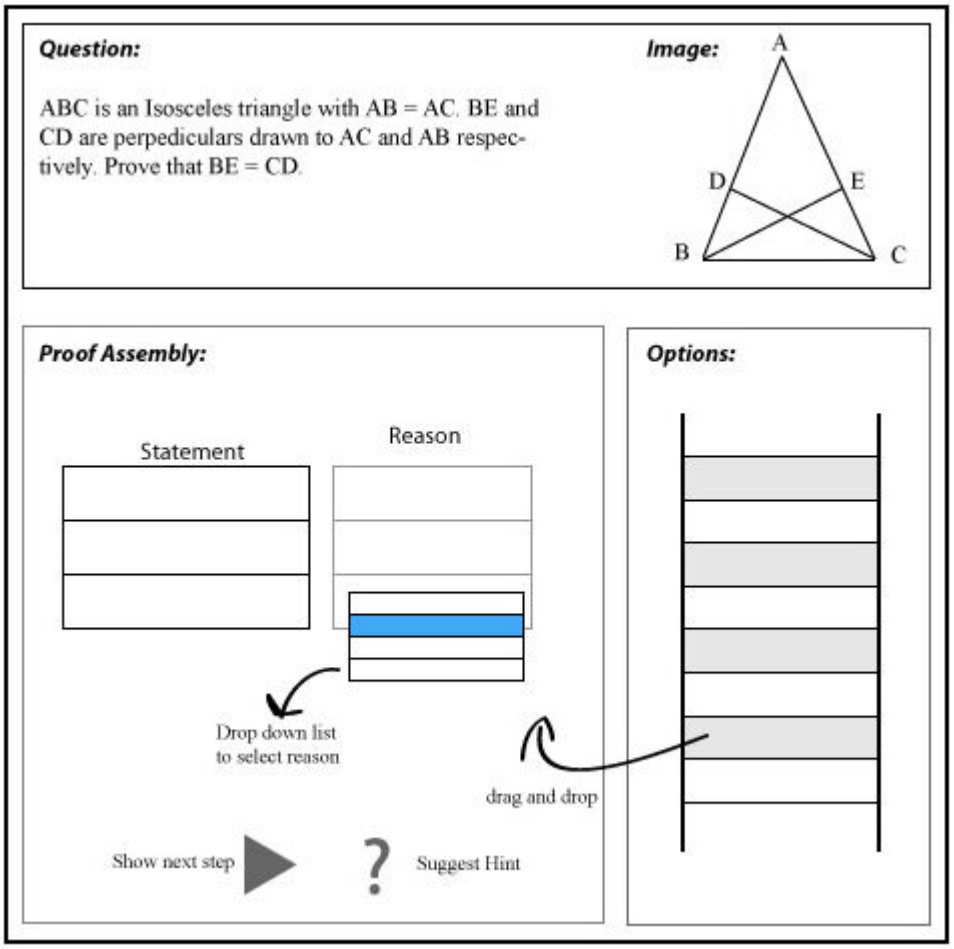


Figure 4.2: A wireframe of the student's interface

step whenever she wants. If the student makes a mistake anywhere, the system will give appropriate feedback along with hints.

4.2 Content Creation Module

As the name suggests, this module is for the content creator to enter questions and the model solutions into the system. A question can have any number of solutions. A question can consist of both text and images. The solution is entered in the form of a tree using the content creation interface which is discussed in the subsequent sections. Once all the solutions are entered into the system, the module will export the solution into an XML format which is processed subsequently by the Solution

Tree Module.

4.2.1 The Solution Tree

The solution tree consists of "nodes" which are connected by "links". Nodes are of four types. Hypothesis nodes (G-node) to represent standard results, theorems and other statements that are provided into the hypothesis of the question. Implication nodes (I-node) to represent statements that are inferred from other G-nodes and I-nodes. Hint nodes (H-node) to save problem specific hints for the question and Dummy nodes (D-node) for storing the related wrong options which are either manually entered by the content creator or automatically generated as extra options. Each node consists of an *Id* to uniquely identify the node, a *type* to denote which type of node it is, a *statement* and a *reason*.

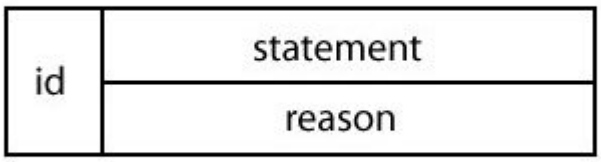


Figure 4.3: Structure of an I-node

The structure of an I-node is shown in figure 4.3. The structure of G-node and D-node are similar except that the *reason* field is by default marked as "Given" and "Dummy" respectively. H-nodes are a little different from the other three because imposing a structure on the hints would mean restricting their effectiveness. Keeping this in mind, the hint node has just an *Id*, a *type* and an *html* field in which any valid html can be written. This will ensure ensure the flexibity in the structure of the hints. With this hints can be anything from simple text explanations to embedded animations or youtube videos.

Fig. 4.4 shows an example each kind of node. The node is with $id = 1$ is a G-node and is represented by a rounded cornered rectangle. The one with $id = 2$ is an I-node and is represented by a rectangle. The D-node with $id- = 3$ is represented with a dotted rectangle while the hint node is represented by a grey rectangle.

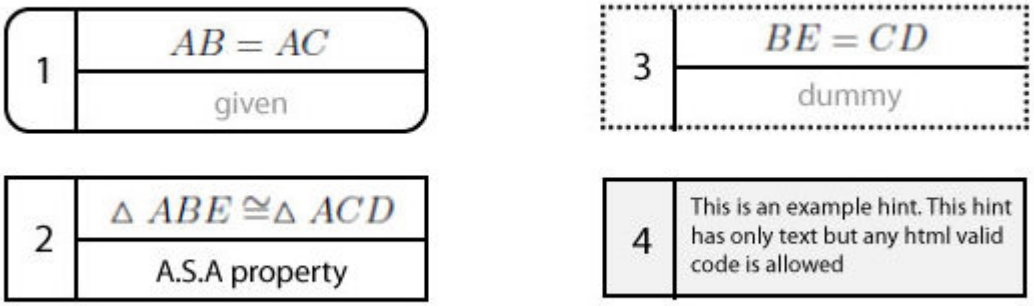


Figure 4.4: Example of each node

4.2.2 The Content Creation Interface

A wireframe of the content creation interface is shown in figure 4.5. The interface consists of six major divisions as shown the functionalities of which are discussed below:

- **Question:** This division is for the content creator to enter the question into the system.

- **Images:** Most questions in Geometry contain figures. The content creator can upload images into the system here. Some problems require the student to make constructions as part of the solution (like dropping a perpendicular) in which case the image that is displayed should be replaced by another image. All the images that are to be used in the course of the solution are uploaded here.

- **Menu:** This menu contains buttons and tools to enable the content creator to construct the solution tree.

- **Entity Directory:** A proof consists of many entities like line segments, angles, triangles etc. All such valid entities are to be defined by the user in this division before using them in the proof. Having a list of all valid entities serves two purposes. It ensures that entities are not duplicated. *i.e.* $\triangle ABC$ and $\triangle BAC$ mathematically mean the same and defining the entity here prevents duplication of entities and helps in maintaining uniformity of notation.

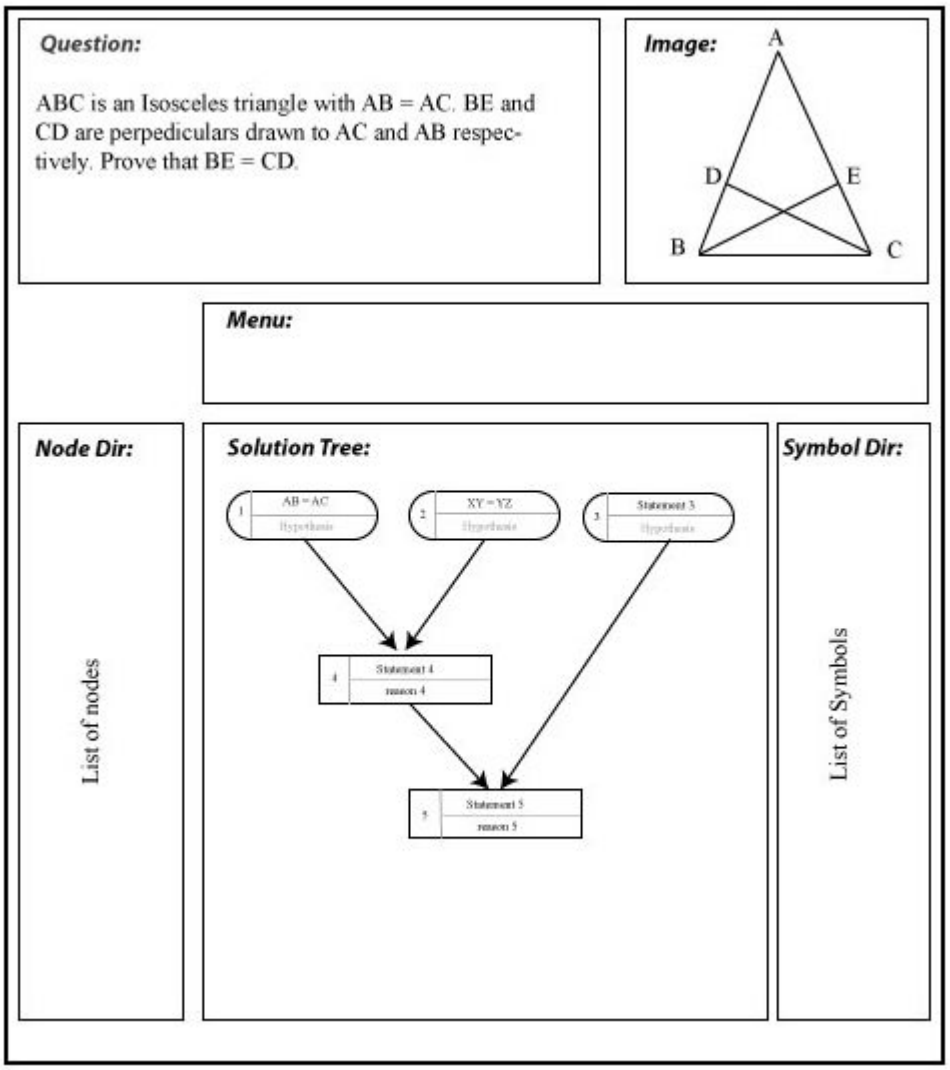


Figure 4.5: A wireframe of the Content Creation Interface

A standard and consistent naming of entities and variables will help make it easier for automating the generation and creation of dummy options. Further it is sometimes convenient to call $\angle ABC$ as just $\angle B$. The entity directory allows the content creator to define such short notations for entities and use them in proofs.

- **Solution Tree** The Solution Tree can be constructed with the help of the tools provided in the menu. A question can have any number of solutions. The content creator has to create a new tree for each solution.

- **Node Directory:** Like we have seen in Example 1 and Example 2 of section 3.3, a single statement might appear in different different solutions. The node directory provides a quick reference for all the nodes that have been entered so far so that they can be easily reused thereby avoiding repetitive typing.

4.2.3 The XML Tree

Once the content creator enters the question and constructs the solution trees, the information should be saved for further use. All the information related to the problem are saved and exported as XML. The platform independent nature of XML and the availability of XML parsers for most programming languages makes it an ideal choice for providing the required modularity for the system. If the interface needs to be changed in the future or to be coded in a different language, it can easily be done as long as the new module also adheres to this XML schema.

4.3 The Solution Tree Module

Different solutions of the same problem invariably have a lot of common nodes which are repeated in the XML (CCXML) generated by the CC module. The idea is to remove redundancies in the XML representation, merge individual solution trees along the common nodes, generate extra Dummy Nodes (if required) and form a General Solution Tree (GST) which the Solution Matching Module can use to match the solution that is entered by the student to one of the solutions entered by the content creator. The Solution Tree Module generates GST from the CCXML.

4.3.1 The Equation Node

The Equation Node is the fundamental element of the GST. An Equation Node should contain all the information present in solution tree node. In addition to that the Equation Node should also have the flexibity to act as a hinge node (a node common to two or more solutions) along which two solutions are merged. The datastructure must be implemented in such a way that no information is lost when merging two occurrences of the same statement in different solutions. A detailed implementation of the Equation Node is described in section 5.2.1

4.3.2 Merging Solutions

The first step towards generating the GST is to merge the independent solution trees. The solution trees are merged along the nodes that have the same statement. Merging is done by identifying the hinges *i.e.* the nodes that are common to two or more solutions. Once the hinges are identified, all the links associated with hinge nodes must be categorized per solution and rewritten accordingly. The tree that is generated after merging all the solutions is called Merged Solution Tree (MST).

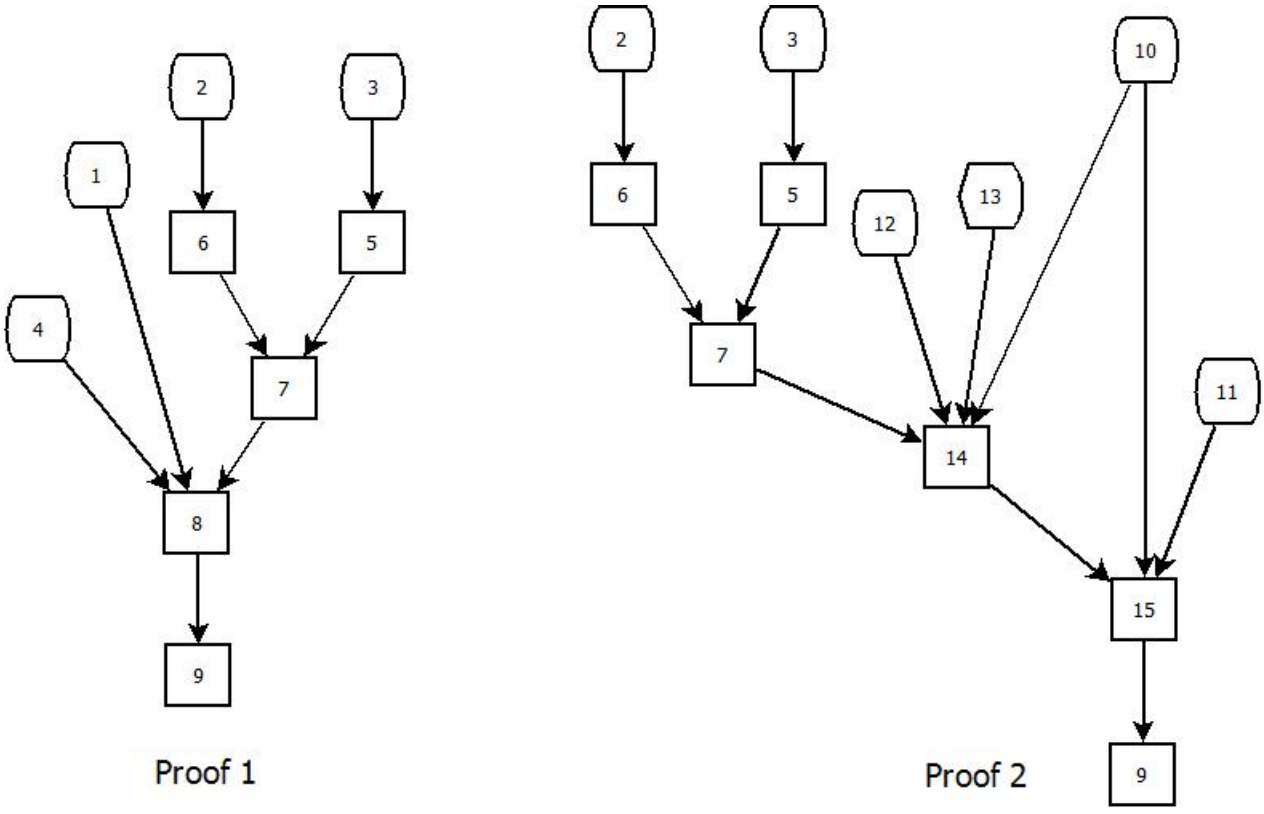


Figure 4.6: P1, P2 of Example 2 represented as solution trees

4.3.3 The General Solution Tree

Once the MST is created, additional Dummy Nodes which are not part of the solution but are misleading options based on commmon student misconceptions are created and attached to the MST at appropriate nodes. Dummy Nodes can also be added by the CC while constructing the solution tree. Automatic generation of Dummy Nodes is not a trivial task and requires considerable intelligence and expert knowledge coded

into it's algorithm. The MST with the Dummy Nodes attached is called the General Solution Tree (GST).

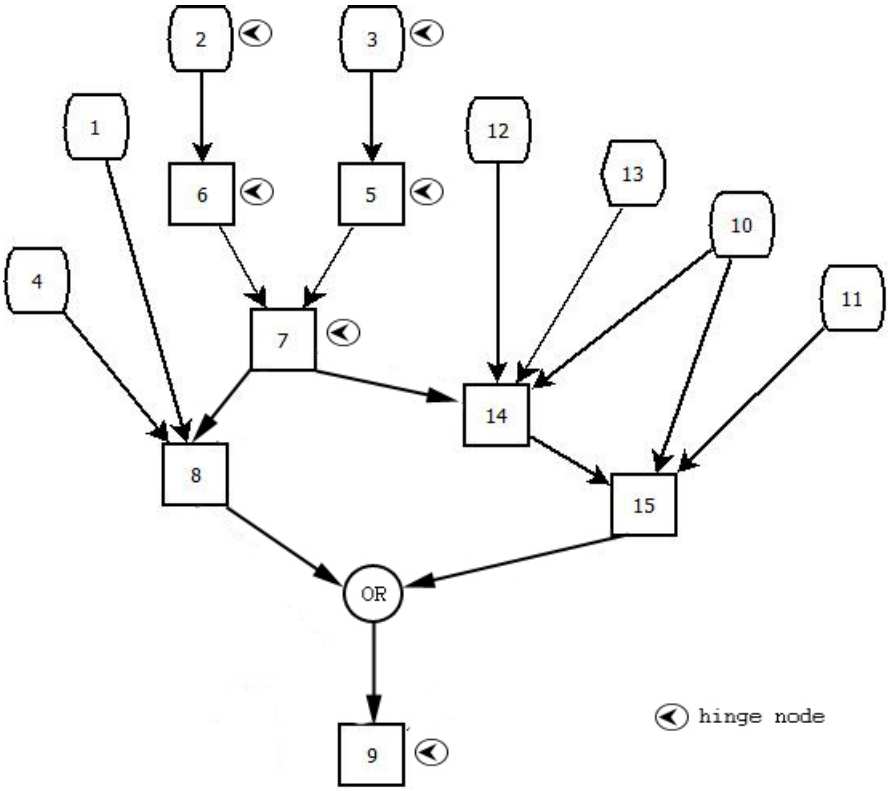


Figure 4.7: GST formed after merging P1 and P2 of Example 2

4.4 Solution Matching Module

The Solution Matching module reacts to what the student is doing. It gets the input from the PA Module and then gives the appropriate output to the student. The student makes an assertion and gives a reason for it. By traversing through the GST, the Solution Matching Module determines if that particular step is allowed as the next step. If it is correct, then it adds that statement to the proof, refreshes the Assertion List and prepares for the next step. Otherwise, it calls the Hint Generation Module.

4.5 Hint Generation Module

The Hint Generation is responsible for giving appropriate feedback when the student makes a mistake or presses the hint button or the "show next step" button.

There are four ways in which HG module can be invoked:

- **The Assertion is wrong**: In this case, if the mistake is a known misconception and a related Hint Node exists in the GST, then the contents of that Hint Node are displayed. Otherwise a new hint is constructed based on the next steps of the solution.
- **The Assertion is correct but the reason is wrong**: In this case one good hint could be to pop out the definition of the wrong option selected. If the mistake is a known misconception, then the system should take appropriate action.
- **The student presses the hint button**: In this case, the system can look at the next possible steps, pick one of the nodes that is not a G-node and construct a hint based on its parents.
- **The student presses the show next step button**: In this case, the system can look at the next possible steps, pick one of the nodes that is not a G-node and show the next step.

Developing algorithms for hint generation is an open ended problem and there can be a number of ways to approach it. Some of them are discussed in Chapter 6.

Chapter 5

Implementation

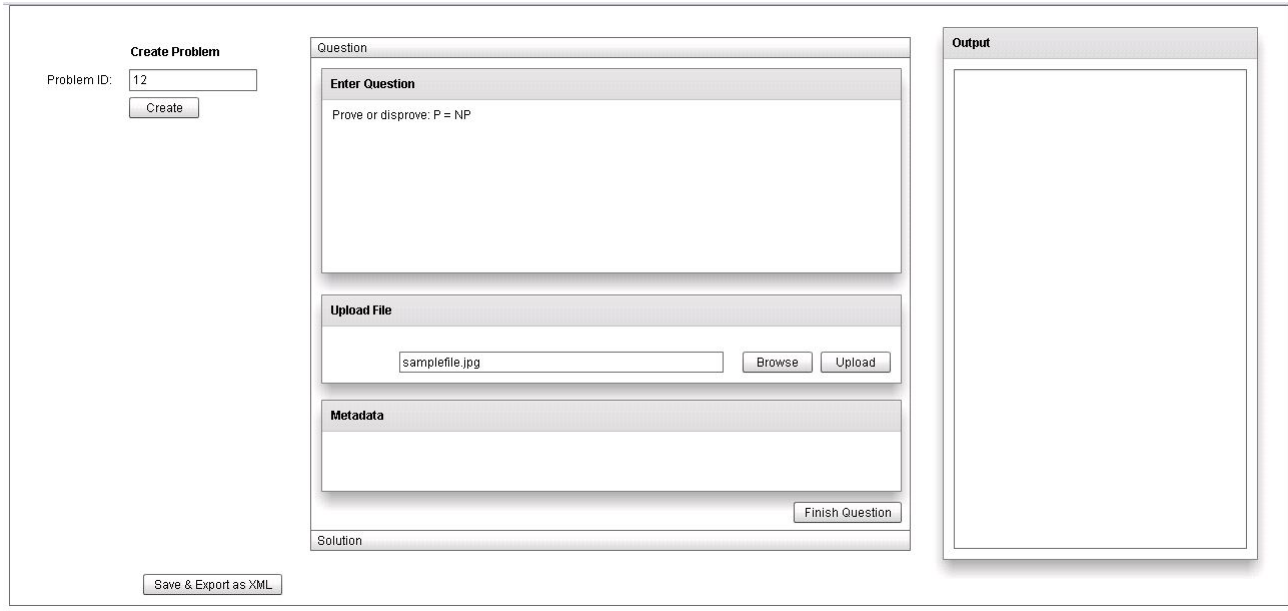
The system was designed taking into account all aspects of the system. The implementation however is limited to the core functionalities of the system only. The Content Creator Interface has been simplified to a form based input and intelligent hint generation is not implemented due to various constraints. The project is implemented using Adobe Flex 4.0 SDK which is based on the Adobe Flash platform and PHP for file handling.

5.1 The Content Creation Module

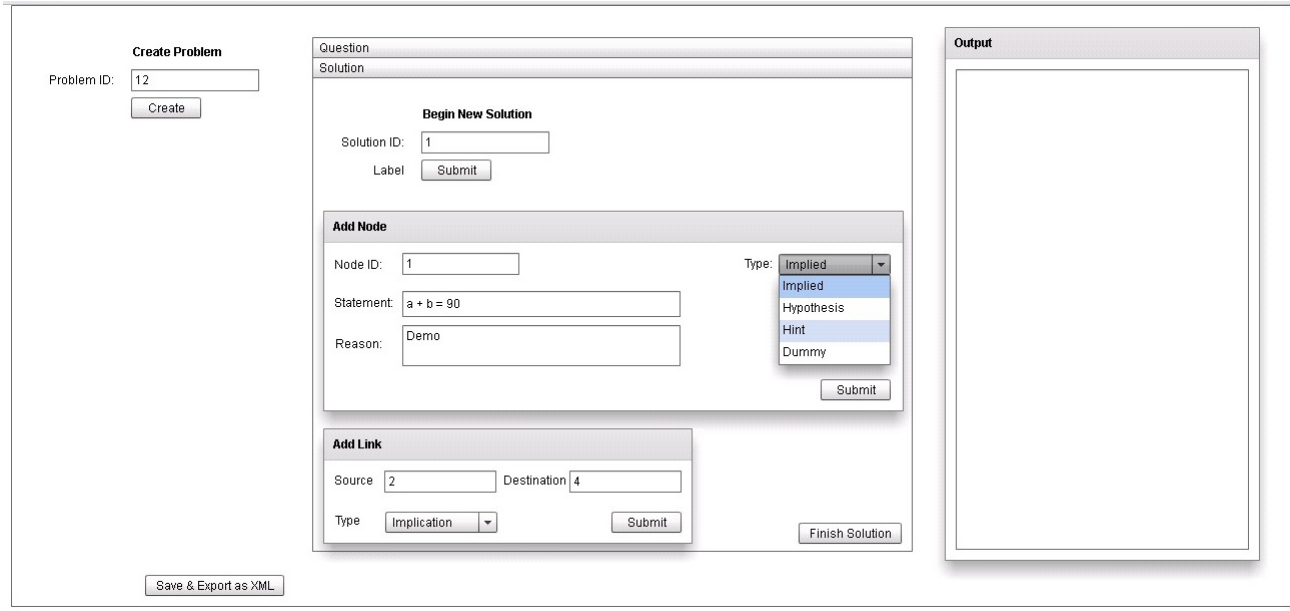
5.1.1 The Content Creation Interface

Since the project is focused on developing the core architecture of the system, the CC Interface has been vastly simplified into a form-based input with an accordion for constructng the tree rather the proposed interactive GUI as described in chapter 4. The content creator first creates a problem by specifying the problem id. Once the problem is created, the question details can be entered using the 'Question' interface as shown in Fig. 5.1

To enter the solution, the user can switch the accordion to 'Solution' mode, create a new solution and enter nodes and links one by one. The confirmation of each action will be visible in the output frame on the right. The user can let the system know that the solution has been complete by clicking the 'Finish Solution' button. The user can now begin a new solution by repeating the process or save and export the solution by clicking on the 'Save and Export' button.



(a) *Screenshot of the Content Creation Interface in 'Question' mode*



(b) *Screenshot of the Content Creation Interface in 'Solution' mode*

Figure 5.1: Screenshots of the Content Creation Interface

5.1.2 The XML Tree

As mentioned in section 4.2.3, the solution tree once created by the content creator is saved in XML format. The XML grammar has been carefully designed to be intuitive and generic enough to accommodate extensions of the system to handle proof type problems in all topics in mathematics. This XML model is described below.

- Operands are defined between their respective tags. For example $\triangle ABC$ and line segment AB are represented as:

```
<tri>ABC</tri>
<lineseg>AB</lineseg>
```

- **Operators** like $=$, $+$, $-$ etc. have their own tag defined. The operands are written as the children on the operator in the XML. All the operators are ranked according to their priority in evaluation. Brackets are also considered as operators in this design. The equation $AB = AC$ is represented as:

```
<eq>
  <lineseg>AB</lineseg>
  <lineseg>AC</lineseg>
</eq>
```

- **Statement** is constructed using operator and operands and is contained within the <statement >. For example, the statement $\angle ABC + \angle BAC = 90$ is represented as:

```
<statement>
  <eq>
    <plus>
      <ang>ABC</ang>
      <ang>BAC</ang>
    </plus>
    <num>90</num>
  </eq>
</statement>
```

- **Node** is most fundamental unit of the Solution Tree. The id, type, statement text, parsed statement and reason are saved in respective tags as shown below.

```
<node id="2" type="g-node">
  <text>BDC = 90</text>
  <statement>
    <eq>
     <ang>BDC</ang>
      <num>90</num>
    </eq>
  </statement>
  <reason>given</reason>
</node>
```

- **Link** can be formed between two any nodes. The linking is done using node IDs. Suppose we have two nodes n_1 and n_2 with node IDs "4" & "6" respectively and $n_1 \Rightarrow n_2$. This link is represented as:

```
<link type="implication" source="4" target="6" />
```

  Suppose there is another node n_3 with node ID "5" such that $n_1,\ n_3 \Rightarrow n_2$. Then these two links are represented as:

```
<link type="implication" source="4" target="6" />
<link type="implication" source="5" target="6" />
```

- Each **solution** is represented as a collection of **nodes** and **links**. The structure of a solution with $id\ =\ 2$ is as follows:

```
<solution id="2">
  <node id="1" type="g-node">
    <text>AB = AC</text>
    <statement>
      <eq>
  <lineseg>AB</lineseg>
```

```
  <lineseg>AC</lineseg>
      </eq>
    </statement>
 <reason>given</reason>
  </node>

  ...
  ...
  ...

  <node id="8" type="i-node">
    <text>\vartriangle ABE \cong \vartriangle ACD</text>
    <statement>
      <cong>
       <tri>ABE</tri>
        <tri>ACD</tri>
      </cong>
    </statement>
    <reason>ASA property</reason>
  </node>

  <link type="implication" source="1" target="3" />
  <link type="implication" source="2" target="5" />
  ...
  ...
  ...
  <link type="implication" source="4" target="8" />
</solution>
```

- Each **problem** is saved as a different XML file. The structure of the saved file is as follows.

```
<problem id="2">
  <question>
    Contains the text of the question as entered by the
```

```
      content creator
    </question>

    <image src="path/to/image" />
    <image src="path/to/second/image">

    <solution id="1">
    ...
    ...
    ...
    </solution>
    ...
    ...
    ...
    <solution id="n">
      ...
      ...
      ...
    </solution>

  </problem>
```

5.2 The Solution Tree Module

The Solution Tree Module consists of two parts. In the first part, the CCXML is parsed and all the nodes are converted into the *Equation* datastructure. In the second part, different solution trees are merged to form the GST which is then saved and exported as an XML file.

5.2.1 The Equation Datastructure

The Equation Node is the basic element of the General Solution Tree and is defined by the *Equation* datastructure. Each *Equation* node has an *id* to uniquely identify the node, a *stmtText* which contains the statement (verbatim) as entered by the CC and a *stmtXML* which is the *stmtText* parsed into XML format in the CC Module.

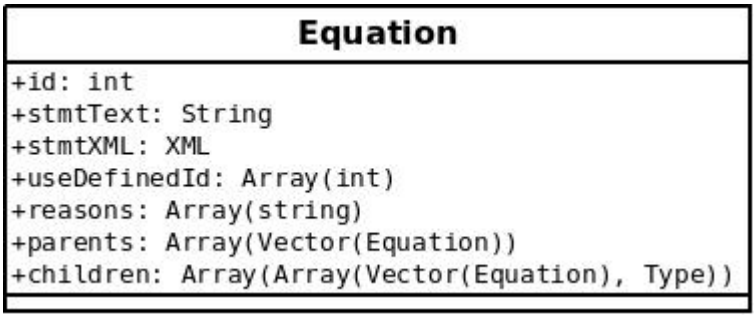


Figure 5.2: Equation Class and it's Attributes

Any two nodes in the original XML which have the same *stmtText* are merged into the same *Equation* node. Their original user defined node Ids are stored in the array *userDefinedId* where the array is indexed by the solution id.

A statement may be repeated in two different solutions of the same problem but it is not necessary that they need to have the same reason or the parents. Hence the reasons for each statement are also in an Array (indexed by solution id) in *reasons*.

The children of each node are saved in *children* which is a three-dimensional array with solution id, Vector of all children and the type of nature of link connecting to each child as it's three dimensions. *parents* is an array of vectors indexed by solution id which contain the parents of the node.

5.2.2 Merging Solutions and generating the GST

The CCXML is first parsed and each node is converted to an *Equation* node. All the generated Equation Nodes are pushed into a Vector (*stmtVec*). When a new node is parsed, we check if an Equation Node with the same statement exists in the statement vector. If it exists, then the contents of this node are merged with the node that is already present in the statement vector. If it doesn't exist, a new Equation is created from the node. The *children* and *parents* arrays in Equation Node are populated by parsing the links in CCXML. The working is illustrated by Algorithm 1.

The design of the Equation datastructure ensures there is no loss of data during the merging. Once GST is created, it is exported and saved as an XML file. The XML format is as follows.

```
<problem id="2">
```

foreach *node in Solution Tree* **do**
  if *Equation with same Statement as node exists* **then**
    Merge contents of *node* with *Equation*
  else
    Create new *Equation* with contents of *node*
  end
end
foreach *link in Solution Tree* **do**
  update *children* and *parents* arrays of corresponding *Equations*
end

Algorithm 1: Merging Solutions

```
<question>
  Contains the text of the question as entered by the
  content creator
</question>
<image src="path/to/image" />
<image src="path/to/second/image">
<equations>
  <equation id="$eqn_id">
    <text>AB = AC</text>
    <statement>
      <eq>
        <lineseg>AB</lineseg>
        <lineseg>AC</lineseg>
      </eq>
    </statement>
  </equation>
  ...
  ...
  ...
</equations>
<solution id="1">
  <link src="4" target="7" type="implication" />
  ...
  ...
  ...
```

```
    <link src="11" target="14" type="implication" />

    <reason id="$eqn_id">Given</reason>
    ...
    ...
    ...
    <reason id="$eqn_id">Given</reason>
  </solution>
  ...
  ...
  ...
</problem>
```

5.3 Solution Matching Module

The Solution Matching reacts to the student's actions. First, the XML file which contains the GST is parsed into a tree of *Equation* Nodes. The SM module gets it's input from the Proof Assembly Module. If the assertion and reason that are entered by the student are correct, it accepts the solution and moves on to the next step. If there is any mistake in any of the steps, it invokes the Hint Generation module. The GST lies at the heart of the SM module. The working of the SM Module is explained through the following definitions and pseudocode.

- **children(n, GST):** This function takes a node n and GST as input and returns an array of all the children of node n.
- **parents(n, GST):** This function takes a node n and GST as input and returns an array of all the parents of node n.
- **refreshAssertionStack():** Refreshes the assertion stack to bring new options.
- **entered_list:** List of all nodes that have been entered as part of the solution.
- **allowed_list:** List of all nodes that are valid as a next step.
- **refreshAllowedList(n):** This function refreshes the allowed list when node n is added to the entered_list. The function is defined in Algorithm 2.

```
refreshAllowedList(n)
foreach child m of n;
do
    if all parents of m are in entered_list then
        add m to allowed_list;
    end
end
```

Algorithm 2: Refresh Allowed List

- **Working:** When the child enters a new assertion along with the reason, the solution matching module has to check if the assertion is valid or not. We maintain two lists, entered_list and allowed_list. At the beginning, the entered_list is empty and all the hypothesis nodes are in the allowed_list. Suppose a node n is entered by the student:

```
if assertion AND reason are correct then
    add node n to entered_list;
    refreshAllowed(n);
    refreshAssertionStack();
else
    callHintGenerationModule();
end
```

Algorithm 3: Working of SM Module

5.4 Proof Assembly Module

The Proof Assembly Module is the interface that the student uses to construct the proof. A screenshot of the interface is shown in 5.3. The question is shown in the Question division along with any images present. The student has to choose one of the statements from the Assertions on the left, drag and drop it in the Proof area using the mouse. Once the assertion is dropped, the "Select Reason"division will show four options from which the student has to choose the correct reason for the statement. Once the correct reason is selected, the assertion will be accepted. Otherwise appropriate feedback will be given. The student can also ask for a hint or ask for the next step using the Help menu in the right bottom corner.

The hint generation module has not been implemented. All the hints shown in the demo are inputted by the content creator as hint nodes through the content creation

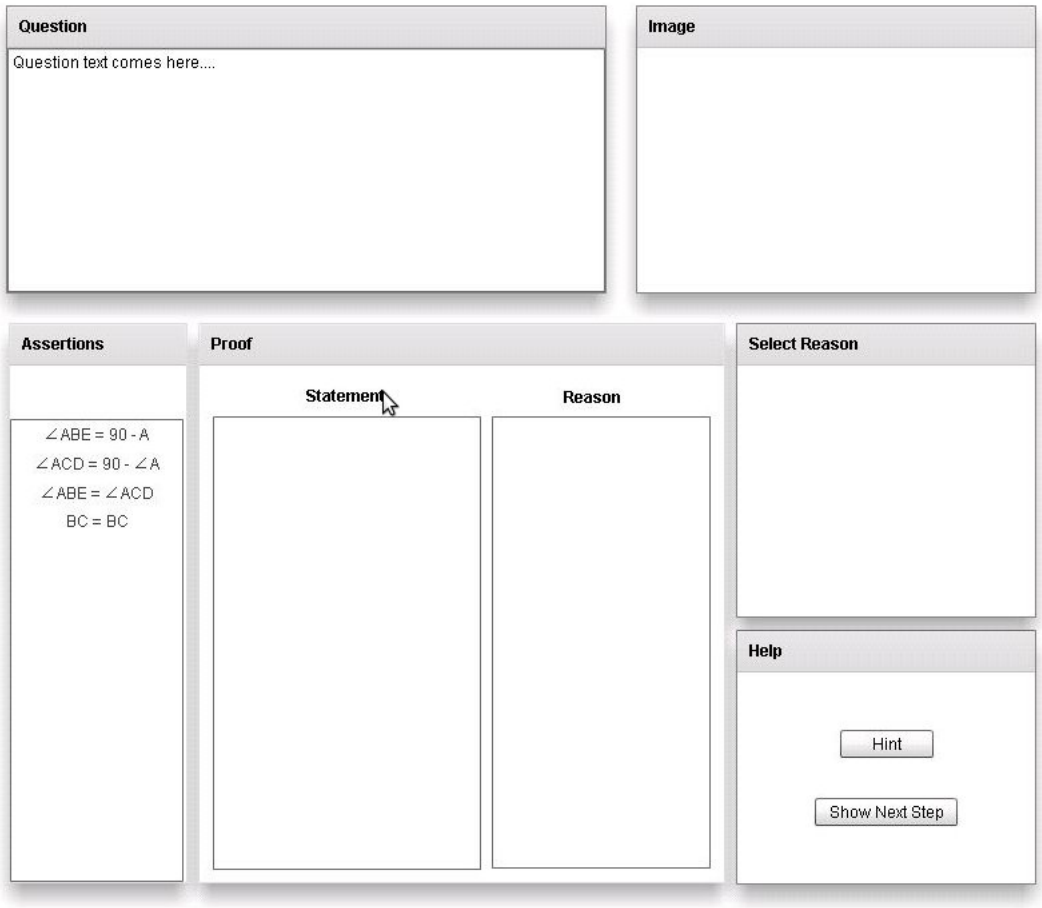


Figure 5.3: Screenshot of the Proof Assembly interface

interface.

Chapter 6

Conclusion and Future Work

In chapter 3, it is mentioned that the ultimate goal of the project is to have a system in which students can solve proof type problems and get instant feedback and hints in a way that is as intuitive as writing on paper. This project is only a first step towards achieving that goal. The modular architecture ensures that each module can be extended and developed independently. This opens up a lot of avenues for future research and development. The Hint Generation Module and the Content Creation Module have ample of scope for improvement.

6.1 Interfaces and Hint Generation

The Content Creation Interface needs to be developed on the lines of the design described in 4.2.2. The interfaces are built only for the prototype. More effort is required towards creating a flawless user experience for both PA interface and CC interface.

Automatic hint generation is not implemented as part of the project. The XML representation of the GST gives the required cross platform compatibility to use logic and functional programming languages like Prolog or Haskell to incorporate artificial intelligence methods for generating hints and dummy options. Further, concepts in Formal methods can be used to build the Solution Matching Module to accept any valid mathematical statement from the student and give feedback on it's correctness. This will eliminate the need for the content creator to manually enter all the possible solutions.

6.2 Evaluation

The system must be evaluated from both a design perspective and from an Educational Technology perspective. Evaluation must be done for each module and also for the whole integrated system. The interfaces must be evaluated for the user experience. As mentioned earlier, a flawless user experience is crucial for the success of the system as an effective learning tool because a bad interface could lead to cognitive overloading of the student's working memory.

The quality of the automatically generated hints, options and dummy nodes must be independently evaluated by a subject expert. Over all, the effectiveness of the system as a learning tool should also be evaluated either in a laboratory setting with a control group or by subject experts.

6.3 Integration

The prototype once finished into a product must be integrated into the Mindspark's system. Additional functionalities like logging student's reactions, meta data for the questions etc. need to be added to the system. Once the system is integrated, a lot of data regarding mistakes that students make, hints students are likely to ask etc. can be easily logged and collected.

Data mining techniques can be used to get valuable insights into common misconceptions the students might have in a particular topic. If these misconceptions are identified, then teachers can address them by taking some extra care while teaching.

6.4 Expanding the Scope

It is important to note that the underlying premises that the system is based on upon are not exclusive to geometry. The system can be extended to other areas of mathematics for the middle school level and for higher classes as well. However, the drag and drop mechanism for solving proofs is suitable primarily for geometry proofs. Proofs in algebra are different from proofs in geometry because the next step is algebra is easy to predict when there are options to choose from. A subjective typing based input would be preferred for proofs in Algebra [3].

Bibliography

[1] William Curtis. *How to Improve Your Math Grades*. Occam Press California, 2538 Milvia St. Berkeley, CA 94704-2611, 2008. chap. 1.

[2] David Foster. *Assessing Mathematical Proficiency*, volume 53. MSRI Publications, 2007. Chap. 12.

[3] Brian Grossman. Intelligent algebraic tutoring based on student misconceptions. Master's thesis, Massachusetts Institute of Technology, 1996.

[4] Adobe Inc. Adobe flex documentation, June 2011. http://www.adobe.com/devnet/flex/documentation.html.

[5] Berinderjeet Kaur. Some common misconceptions in algebra. *Teaching and Learning, 11(2),33-39*, 1990.

[6] Lindsay M. Keazer. Students' misconceptions in middle school mathematics. Master's thesis, Ball State University Muncie, Indiana, 2003.

[7] K. R. Koedinger and A. T Corbett. Cognitive tutors: Technology bringing learning science to the classroom. *The Cambridge Handbook of the Learning Sciences*, 2006.

[8] Roblyer M. and Doering A. *Integrating Educational Technology into Teaching*. Pearson Education, 5th edition, 2009.

[9] M. Matz. Towards a process model for high school algebra errors. *Intelligent tutoring systems (pp. 25-50)*, 1982.

[10] Robert McCormick. Conceptual and procedural knowledge. *International Journal of Technology and Design Education*, 7:141–159, 1997. 10.1023/A:1008819912213.

[11] http://www.mindspark.in Mindspark.

[12] Alwyn Olivier. Handling pupils' misconceptions. *Mathematics Education for Pre-Service and In-Service*, 1992. page 193-209.

[13] Learn Quebec. Algebra: Some common misconceptions, August 2010. Algebra misconceptions with visuals.

[14] J. Ryan and J. Williams. *Mathematics 4-15: learning from errors and misconceptions*. Open University Press Maidenhead, 2007.

[15] World Wide Web Consortium (W3C). www.w3.org/xml, June 2011.

[16] www.counton.org. Misconceptions in mathematics, August 2010. http://www.counton.org/resources/misconceptions/.

[17] www.toptenreviews.com. Algebra software review.

[18] Zhicheng Zhang. Carnegie learning cognitive tutor algebra 1 and geometry follow-up report. 2007.